



Introduction to Microcontrollers Lab Manual

Featuring the PIC24F Family

Note the following details of the code protection feature on Microchip devices:

- Microchip products meet the specification contained in their particular Microchip Data Sheet.
- Microchip believes that its family of products is one of the most secure families of its kind on the market today, when used in the intended manner and under normal conditions.
- There are dishonest and possibly illegal methods used to breach the code protection feature. All of these methods, to our knowledge, require using the Microchip products in a manner outside the operating specifications contained in Microchip's Data Sheets. Most likely, the person doing so is engaged in theft of intellectual property.
- Microchip is willing to work with the customer who is concerned about the integrity of their code.
- Neither Microchip nor any other semiconductor manufacturer can guarantee the security of their code. Code protection does not mean that we are guaranteeing the product as "unbreakable."

Code protection is constantly evolving. We at Microchip are committed to continuously improving the code protection features of our products. Attempts to break Microchip's code protection feature may be a violation of the Digital Millennium Copyright Act. If such acts allow unauthorized access to your software or other copyrighted work, you may have a right to sue for relief under that Act.

Information contained in this publication regarding device applications and the like is provided only for your convenience and may be superseded by updates. It is your responsibility to ensure that your application meets with your specifications. MICROCHIP MAKES NO REPRESENTATIONS OR WARRANTIES OF ANY KIND WHETHER EXPRESS OR IMPLIED, WRITTEN OR ORAL, STATUTORY OR OTHERWISE, RELATED TO THE INFORMATION, INCLUDING BUT NOT LIMITED TO ITS CONDITION, QUALITY, PERFORMANCE, MERCHANTABILITY OR FITNESS FOR PURPOSE. Microchip disclaims all liability arising from this information and its use. Use of Microchip devices in life support and/or safety applications is entirely at the buyer's risk, and the buyer agrees to defend, indemnify and hold harmless Microchip from any and all damages, claims, suits, or expenses resulting from such use. No licenses are conveyed, implicitly or otherwise, under any Microchip intellectual property rights.

Trademarks

The Microchip name and logo, the Microchip logo, dsPIC, KEELOQ, KEELOQ logo, MPLAB, PIC, PICmicro, PICSTART, PIC³² logo, rfPIC and UNI/O are registered trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.


FilterLab, Hampshire, HI-TECH C, Linear Active Thermistor, MXDEV, MXLAB, SEEVAL and The Embedded Control Solutions Company are registered trademarks of Microchip Technology Incorporated in the U.S.A.

Analog-for-the-Digital Age, Application Maestro, chipKIT, chipKIT logo, CodeGuard, dsPICDEM, dsPICDEM.net, dsPICworks, dsSPEAK, ECAN, ECONOMONITOR, FanSense, HI-TIDE, In-Circuit Serial Programming, ICSP, Mindi, MiWi, MPASM, MPLAB Certified logo, MPLIB, MPLINK, mTouch, Omniscient Code Generation, PICC, PICC-18, PICDEM, PICDEM.net, PICkit, PICtail, REAL ICE, rfLAB, Select Mode, Total Endurance, TSHARC, UniWinDriver, WiperLock and ZENA are trademarks of Microchip Technology Incorporated in the U.S.A. and other countries.

SQTP is a service mark of Microchip Technology Incorporated in the U.S.A.

All other trademarks mentioned herein are property of their respective companies.

© 2011, Microchip Technology Incorporated, Printed in the U.S.A., All Rights Reserved.

 Printed on recycled paper.

ISBN: 978-1-61341-251-0

Microchip received ISO/TS-16949:2002 certification for its worldwide headquarters, design and wafer fabrication facilities in Chandler and Tempe, Arizona; Gresham, Oregon and design centers in California and India. The Company's quality system processes and procedures are for its PIC[®] MCUs and dsPIC[®] DSCs, KEELOQ[®] code hopping devices, Serial EEPROMs, microperipherals, nonvolatile memory and analog products. In addition, Microchip's quality system for the design and manufacture of development systems is ISO 9001:2000 certified.

QUALITY MANAGEMENT SYSTEM
CERTIFIED BY DNV
== ISO/TS 16949:2009 ==

Table of Contents

Lab 0. Introduction to Microcontrollers	
0.1 Objective	7
0.2 Pre-Lab	7
0.3 The Lab	9
Lab 1. Timers	
1.1 Objective	13
1.2 Pre-Lab	13
Lab 2. Interrupts	
2.1 Interrupts	19
2.2 Pre-Lab	19
2.3 The Lab	23
Lab 3. UART	
3.1 Objective	25
3.2 Pre-Lab	25
3.3 The Lab	30
Lab 4. Watchdog Timer (WDT)	
4.1 Objective	33
4.2 Pre-Lab	33
4.3 The Lab	35
Lab 5. Comparator	
5.1 Objective	37
5.2 Pre-Lab	37
5.3 The Lab	40
Lab 6. Analog-to-Digital Converter	
6.1 Objective	43
6.2 Pre-Lab	43
6.3 The Lab	47
Lab 7. Pulse-Width Modulator (PWM)	
7.1 Objective	51
7.2 Pre-Lab	51
7.3 The Lab	54
Lab 8. External Memory	
8.1 Objective	57
8.2 Pre-Lab	57
8.3 The Lab	60

Introduction to Microcontrollers Lab Manual

Lab 9. Power Management

9.1 Objective	63
9.2 Pre-Lab	63
9.3 The Lab	67

Lab 10. Project

10.1 Objective	69
10.2 The Lab	69

Appendix A. Creating an MPLAB IDE Project

A.1 Step One: Device Selection	71
A.2 Step Two: Language Toolsuite	72
A.3 Step Three: Create New Project File	73
A.4 Step Four: Add Existing Project Files	73

Lab Manual Introduction

INTRODUCTION

The purpose of these labs is to introduce the basic concepts of designing microcontroller based systems. Each lab will introduce a new concept and provide a starting template that will guide the student toward creating a successful project.

Each lab includes a pre-lab section that the student should study prior to entering the lab. It includes a basic description of the topics under consideration, as well as links to many other resources that can be used for further study.

DEVELOPMENT ENVIRONMENT

This document describes how to use one of the starter kits as a development tool to emulate and debug firmware on a target board.

The MPLAB[®] Integrated Development Environment (IDE) is a free, integrated toolset for the development of embedded applications employing Microchip's PIC[®] MCUs and dsPIC[®] DSCs. MPLAB IDE runs as a 32-bit application on MS Windows[®] (XP, Vista, or Windows 7), and is an easy-to-use tool that includes a host of free software components for fast application development and super-charged debugging. MPLAB IDE also serves as a single, unified graphical user interface for additional Microchip and third party software and hardware development tools. Moving between tools is easy, and upgrading from the free software simulator to hardware debug and programming tools is done quickly because MPLAB IDE has the same user interface for all tools.

These labs will use Microchip MPLAB IDE v8.63. It is available from the lab instructor or as a free download from www.microchip.com. Future versions may work, but the exact instructions may no longer be accurate.

Each lab will include a project template. This template will include all the settings required for a suitable solution on the target hardware. There are many different ways to solve each problem. Comments in the code like:

```
;  
; CODE GOES HERE  
;
```

will guide the student to locations where code will typically need to be added for a successful outcome. Motivated students may choose to ignore those comments and come up with a different solution to the same problem.

Introduction to Microcontrollers Lab Manual

DEVELOPMENT HARDWARE

These labs are based on the MX Module series manufactured by Stratford Digital. The MX Series includes modules from across all Microchip embedded controller families, and provides a common mechanical and electrical interface. The modules are suitable for a rapid-development environment, but are also engineered to be used in a production environment.

The MX PIC24F EDU Module (STR001) is specifically designed for this lab. The main processor on this module is the Microchip PIC24FJ256GB110. This 16-bit controller has 256 Kb of internal flash memory and many useful peripheral modules. Each lab will explore various aspects of this controller.

The MX PIC24F EDU Module also includes the PICKit™ on-board circuit that allows the board to connect to MPLAB IDE with only a standard USB cable. The debugging and programming resources normally found in an external PICKit 3, MPLAB ICD 3, or MPLAB REAL ICE™ in-circuit emulator are included on board for an optimal development environment.

All labs require the MX PIC24F EDU Module to be inserted into the MX Educational Target Board (STR002).

The MX Educational Target Board provides the power and I/O required by the MX PIC24F EDU Module. This board connects the various I/Os of the module to easy-to-use interface connectors including RS232 DB9 connectors, a breadboarding area, and common headers for other electrical busses.

The MX Series is intended to provide an infrastructure that will allow the user to quickly develop their application on a known-good hardware platform. The commercial modules come with on-board Ethernet and USB (device, host, and on-the-go), as well as various options for external memory. In addition, template projects are provided for MPLAB IDE that include fully working TCP/IP and USB stacks along with demo applications. Only the custom firmware required for the specific application needs to be added.

Note: Although the labs presented here are based on the MX Module series manufactured by Stratford Digital, they can easily be implemented on a variety of other development platforms from Microchip and third parties with minor modifications. Many of the labs are also applicable to other 16-bit Microchip products, including the PIC24H, dsPIC30F and dsPIC33F. For more information, visit www.microchip.com/academic.

Lab 0. Introduction to Microcontrollers

0.1 OBJECTIVE

This lab is an introduction to the following concepts required to complete the labs:

- MX PIC24F EDU Module
- Microchip PIC24FJ256GB110 MCU
- 16-bit MCU Assembly Instruction Set
- Microchip MPLAB IDE Software v8.63

0.2 PRE-LAB

0.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)

0.2.2 Assembly Language

All labs will be written in assembly language.

Assembly language is a low-level language for programming computers, microprocessors, and microcontrollers. It is specific to a particular device or family of devices and is not typically portable between different processors.

A utility program called an assembler is used to translate assembly language into the target hardware machine code. These labs will use Microchip's MPLAB ASM30 assembler to generate machine code for the PIC24FJ256GB110.

Assembly language uses simple instructions, symbols, and numeric characters usually defined by either the assembler utility or the device hardware manufacturer.

The assembler then performs a translation from the mnemonic assembly statements into hardware specific machine instructions and data. The machine instructions can then be programmed into the controller's memory and executed on the specific hardware.

The "16-bit MCU and DSC Programmer's Reference Manual" (DS70157) from Microchip is a complete list of all assembler instructions for the Microchip 16-bit processor family, including the PIC24FJ256GB110.

A further suggestion from Microchip: Microchip Technology Inc. strongly suggests a `.s` extension for assembly source files. This will enable you to easily use the C compiler driver without having to specify the option to tell the driver that the file should be treated as an assembly file. See the "MPLAB C Compiler for PIC24 MCUs and dsPIC DSCs User's Guide" (DS51284) for more details on the C compiler driver.

0.2.3 Timing

The project initializes the PIC24FJ256GB110 to run from its internal Fast RC oscillator. It configures the use of the internal PLL to create an internal clock running at 32 MHz (Fosc). The architecture of the PIC24 specifies that an instruction cycle (Fcy) is com-

prised of two internal clock cycles, i.e. 16 MHz. This represents an instruction period of 62.5 ns. Most instructions are executed in a single instruction cycle. This means that most instructions will execute in exactly 62.5 ns.

The exception to this is when a conditional test is true or the program counter is changed as a result of the instruction. In these cases, the execution takes two instruction cycles, with the additional instruction cycle(s) executed as a NOP. Common forms of these exceptions are the BRA (unconditional / computed branch), indirect CALL/GOTO, all table reads and writes, and RETURN/RETFIE instructions.

Certain instructions that involve skipping over the subsequent instruction require either two or three cycles if the skip is performed, depending on whether the instruction being skipped is a single-word or two-word instruction. Moreover, double-word moves require two cycles. The double-word instructions execute in two instruction cycles.

0.2.4 Delay Loops

Because the instruction period is so small (62.5 ns), it is often necessary to slow processing down when human interaction is required. In this lab we will blink an LED. If we toggled the LED at anything close to the instruction frequency the human eye would not be able to detect the flashing.

A typical delay loop looks like this:

```

MOV      #0xFFFF, W4
DELAY:  NOP
        DEC      W4, W4
        BRA      NZ, DELAY
    
```

The way this loop works is that the first instruction, MOV #0xFFFF, W4, loads the W4 (this is an arbitrary working register and could be any register other than W15, which is the stack pointer) register with the value 65535, which is the number of times we want to loop. This instruction only gets executed once. The next instruction is the NOP instruction, which is a “no operation” instruction, the purpose of which is to waste a processor cycle. Next the DEC W4, W4 instruction decrements the value in W4, and finally the BRA NZ, DELAY checks if the value in W4 has reached zero. If the value in W4 has not reached zero, a branch is executed and we jump back to the NOP instruction and repeat until the W4 register eventually hits zero. At this point the branch is not taken and we fall out of the loop.

Timing calculation for this delay loop:

Instruction	Cycles
MOV #0xFFFF, W4	1
NOP	1
DEC W4, W4	1
BRA NZ, DELAY	2, if the branch is taken, and 1 if the branch is not taken.

- The MOV instruction is executed once, 1 cycle
- The loop consists of the NOP, DEC, and BRA instructions and gets executed 65535 times. Each time through the loop takes 4 cycles, except for the last time, which only takes 3 cycles since the branch is not taken
- The total delay for this code is 1 cycle + (65535 x 4 cycles) – 1 cycle = 262140 cycles

With a 16 MHz instruction frequency, the loop will take 262140 cycles x 62.5 ns/cycle = 0.01638375 sec to execute fully.

0.2.5 I/O Ports

The ability of the microcontroller to execute code is meaningless unless it can take inputs from its environment and drive outputs to control the environment. The digital I/O port is a primary method of doing just this.

Microchip's PIC microcontrollers organize the digital I/Os into "ports". Each port on the PIC24 is a collection of up to 16 individual I/Os, and they are named PORTA, PORTB, PORTC, etc. These I/Os are easily configurable and can be accessed individually or as a group. For now, we will discuss the ability to use them as digital inputs or digital outputs.

All port pins have at least three registers directly associated with their operation as digital I/Os. The Data Direction register (TRISA, TRISB, TRISC, ...) determines whether the pin is an input or an output. If the data direction bit is a '1', then the pin is an input. If the data direction bit is a '0', then the pin is an output.

Hint:

Think "I" for Input and change "I" (letter I) to "1" (one) in your mind.

Think "O" for Output and change "O" (letter O) to "0" (zero) in your mind.

All port pins are defined as inputs by default.

To write data to a digital output, a write to the appropriate Output Latch Register (LATA, LATB, LATC, ...) is required. To read data from a digital input read the appropriate Input Port Register (PORTA, PORTB, PORTC, ...).

Example code to configure port C bit 2 as output and set it low:

```
BCLR TRISC, #2   Configure port C bit 2 as output
BCLR PORTC, #2   Set port C bit 2 latch low
```

Example code to configure port C bit 0 as input and perform a bit compare on that input:

```
BSET TRISC, #0   Configure port C bit 0 as an input
NOP              Required between a port direction change and port write
BTSS PORTC, #0   Check if port C bit 0 is low
```

One instruction cycle is required between a port direction change or port write operation and a read operation of the same port. Typically, this instruction would be a `NOP`.

0.3 THE LAB

0.3.1 Objective

Program the PIC24 microcontroller to blink an LED at a one second interval (LED on for one second, then off for one second).

0.3.2 Bonus

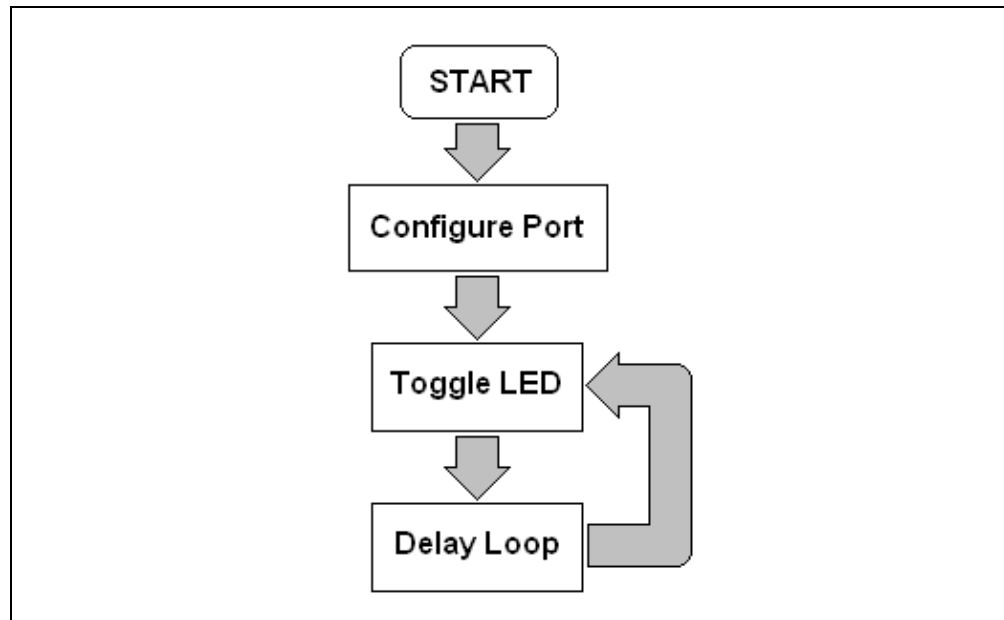
Modify the code so that when one of the push buttons is held, the LED blinks at twice the frequency. You may choose any of the four push buttons. Note the push buttons are mapped to the following ports:

Push Button	PIC® MCU Port
SW1	Port G Bit 12
SW2	Port G Bit 13
SW3	Port G Bit 15
SW4	Port C Bit 1

0.3.3 Pertinent Information

The basic program flow is depicted in [Figure 0-1](#).

FIGURE 0-1: LAB 0 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab0-intro.s`.

As noted in the Pre-Lab, the maximum delay of a delay loop is about 0.016 seconds. Further delaying will be required to obtain the interval of one second. A nested loop (burying one loop within another loop) is one way to accomplish this.

Hint: We advise that you start by writing simple code to initialize the LED port pin and turn the LED on before you dive in and write the delay loop. This will ensure that the project compiles and the device is programmed properly with the simplest code. Debugging of the delay loops will then be confined to just that section of code.

0.3.4 Creating a New Project

See [Appendix A. “Creating an MPLAB IDE Project”](#) for details about using the Project Wizard to create a new project from scratch. The examples show the three files required to add to the project. In order to view the various files in the project, ensure that you have the Project Toolbar enabled. This is done by selecting [View>Project](#) in the menu.

It is also useful to be able to view the output window. This will display various status messages during the process of assembling and linking the project. To enable this window, make sure [View>Output](#) is selected in the menu.

To run the assembler and linker to create the application binary, use the “Make” command. You can access this through the menu [Project>Make](#) or using the shortcut “F10” function key.

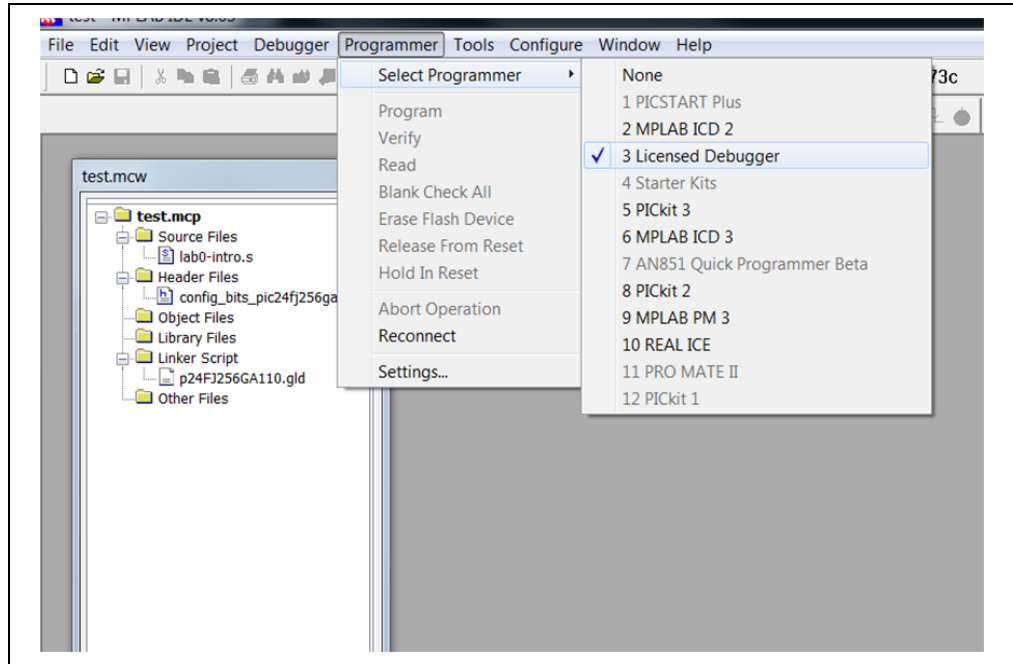
If the make command is successful, the “BUILD SUCCEEDED” message will display at the end of the output window.

0.3.5 Programming the Device

In order to program the device, the build must complete successfully. This will create the binary image that is required to load into the device Flash.

You must configure MPLAB IDE to use the licensed debugger circuit that resides on the MX PIC24F EDU module, see [Figure 0-2](#). In MPLAB IDE, select “Licensed Debugger” from the menu *Programmer>Select Programmer>Licensed Debugger*.

FIGURE 0-2: SELECT PROGRAMMER



Now all that is needed is to program the device. To do this, ensure the programming USB cable is connected, and select *Programmer>Program* from the menu. The output window will display the status of the process. Any errors will be noted in this window.

Once programming is complete, the device should immediately begin to execute the code. If successful, you will see the LED blink at the intended rate.

LABS

NOTES:

Lab 1. Timers

1.1 OBJECTIVE

The lab covers the following concepts:

- PIC24F hardware timer modules
- Assembly language `CALL` and `RCALL` instructions
- Using MPLAB IDE to measure code execution time

1.2 PRE-LAB

1.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Reference Manual – Section 14. Timers (DS39704)

1.2.2 Timers

The PIC24FJ256GB110 device offers five (5) 16-bit timers/counters designated as Timer1, Timer2, ...

With certain exceptions, all of the 16-bit timers have the same functional circuitry. The 16-bit timers are classified into three types to account for their functional differences:

- Timer1 is a Type A timer
- Timer2 and Timer4 are Type B timers
- Timer3 and Timer5 are Type C timers

Type B and Type C timers can be combined to form a 32-bit timer.

Each timer module consists of the following readable/writable registers:

- TMRx: 16-bit Timer Count register
- PRx: 16-bit Timer Period register associated with the timer
- TxCON: 16-bit Timer Control register associated with the timer

Timers can be run from either an external clock connected to the TxCK pin, or the internal clock ($F_{osc}/2$). This lab will use the internal clock as the MX PIC24F EDU Module does not connect an external clock to the TxCLK pins.

1.2.3 Subroutine `CALL`

A subroutine `CALL` in assembly is a powerful tool for reusing code and keeping your code clean and legible. Low-level functions can be abstracted into higher level functions by the use of subroutine. This is similar to the use of a function in the 'C' language. In fact, 'C' functions make use of the `RCALL` assembly instruction.

A `CALL` is similar to a `GOTO` or `BRA` instruction except that it stores the address of the next instruction and allows you to return to it with the `RETURN` instruction.

The 16-bit MCU and DSC device families have two types of calls:

- CALL – Call subroutine
- RCALL – Relative call

When performing function calls, it is recommended that RCALL be used instead of CALL, since RCALL only consumes 1 word of program memory. Table 1-1 highlights some of the main differences in the behavior of CALL and RCALL in modifying the program flow.

TABLE 1-1: LAB 1 METHODS OF MODIFYING PROGRAM FLOW

Condition/Instruction	PC Modification	Software Stack Usage
Sequential Execution	PC = PC + 2	None
BRA Expr ⁽¹⁾ (Branch Unconditionally)	PC = PC + 2*Slit16	None
BRA Condition, Expr ⁽¹⁾ (Branch Conditionally)	PC = PC + 2 (condition false) PC = PC + 2*Slit16 (condition true)	None
CALL Expr ⁽¹⁾ (Call Subroutine)	PC = lit23	PC + 4 is PUSHed on the stack ⁽²⁾
CALL Wn (Call Subroutine Indirect)	PC = Wn	PC + 2 is PUSHed on the stack ⁽²⁾
GOTO Expr ⁽¹⁾ (Unconditional Indirect Jump)	PC = lit23	None
GOTO Wn (Unconditional Indirect Jump)	PC = Wn	None
RCALL Expr ⁽¹⁾ (Relative Call)	PC = PC + 2*Slit16	PC + 2 is PUSHed on the stack ⁽²⁾
RCALL Wn (Computed Relative Call)	PC = PC + 2*Wn	PC + 2 is PUSHed on the stack ⁽²⁾
Exception Handling	PC = address of the exception handler (read from vector table)	PC + 2 is PUSHed on the stack ⁽³⁾
PC = Target REPEAT instruction (REPEAT Looping)	PC not modified (if REPEAT active)	None

- Note 1:** For BRA, CALL and GOTO, the Expr may be a label, absolute address or expression, which is resolved by the linker to a 16-bit or 23-bit value (Slit16 or lit23).
- 2:** After CALL or RCALL is executed, RETURN or RETLW will POP the Top-of-Stack (TOS) back into the PC.
- 3:** After an exception is processed, RETFIE will POP the Top-of-Stack (TOS) back into the PC.

1.2.4 IDE Concepts: Stopwatch

The Stopwatch is useful for simple timing between program halts to time the execution of sections of code.

Several debug tools have a stopwatch feature including.

- MPLAB SIM Simulator: *Debugger>Stopwatch*
- MPLAB ICD 3 in-circuit debugger: *Debugger>Breakpoints, Stopwatch*

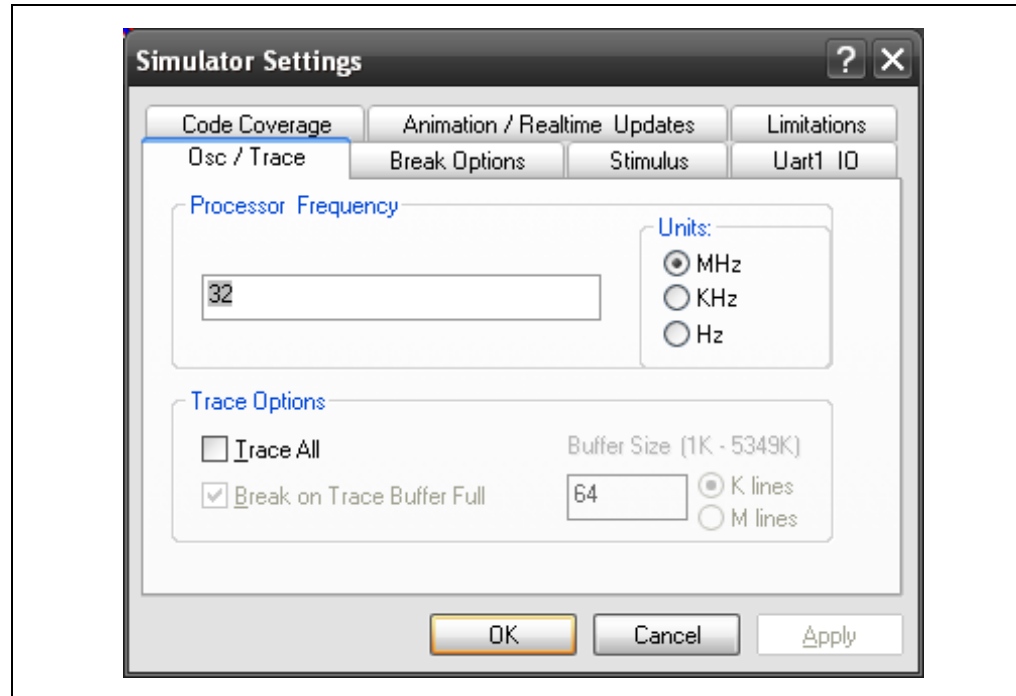
To use the StopWatch with MPLAB SIM Simulator, select *Debugger>Select Tool>4 MPLAB SIM*.

Setup the Simulator processor frequency:

1. Select *Debugger>Settings* and then click on the **Osc/Trace** tab. This is where the clock speed of your hardware is set. These values will also be used to set certain fields in the Stopwatch.
2. Select the units of frequency.
3. Enter the processor frequency value.

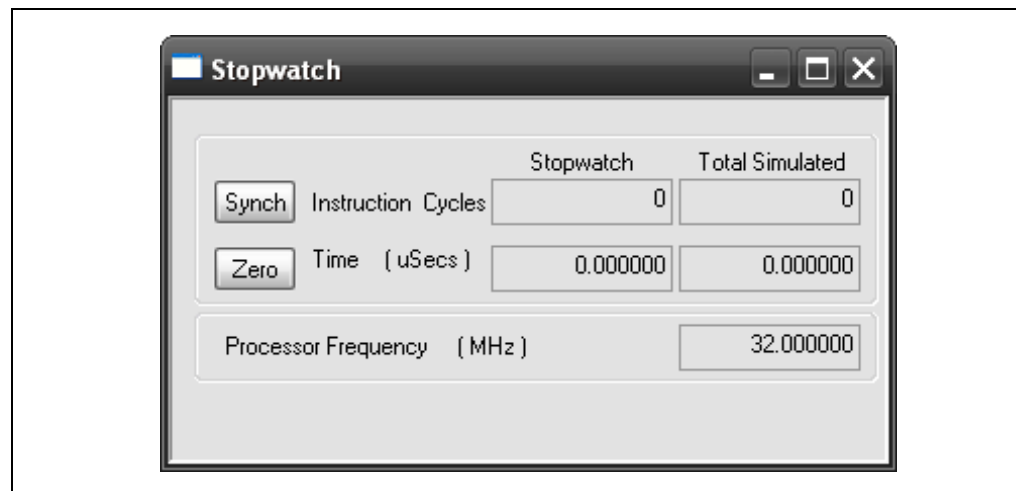
The lab project initializes the PIC24FJ256GB110 to run from its internal Fast RC oscillator and configures the use of the internal PLL to create an internal clock running at 32 MHz.

FIGURE 1-1: LAB 1 STOPWATCH SETUP



Now the Stopwatch is configured. Start the Stopwatch using the menu item. Select *Debugger>Stopwatch*.

FIGURE 1-2: LAB 1 STOPWATCH USE



To use the Stopwatch, insert two breakpoints around the section of code you wish to time. Run the code to the first breakpoint, “Zero” the StopWatch and then run the code to the next breakpoint. The Stopwatch will show the number of instruction cycles and time between the two breakpoints.

Hint: When using the Stopwatch with MPLAB SIM Simulator, it is important to set the Simulator Processor Frequency correctly. Otherwise the Stopwatch time will be incorrect. An easy way to test if this has been set properly is to set breakpoints around a simple instruction such as `MOV`. This should execute in 0.062500 uSecs.

1.2.5 The Lab

1.2.5.1 OBJECTIVE

Program the PIC24 microcontroller to use a hardware timer to blink an LED at a 1 second interval (LED on for 1 second then off for 1 second). Show your calculations for the Timer delay.

Use a subroutine `CALL` or `RCALL` somewhere in the code. A common use would be to move the configuration instructions for the timer into a subroutine. This keeps the main line code cleaner with only one line required to initialize the timer module at the start of the code.

Use the stopwatch to check your timer and verify it is actually running at 1 sec (5 sec for the bonus).

Hint: Use one breakpoint at your LED toggle instruction (`BTG PORTC, #2`), run the code to that breakpoint, then “Zero” the stopwatch and run it again.

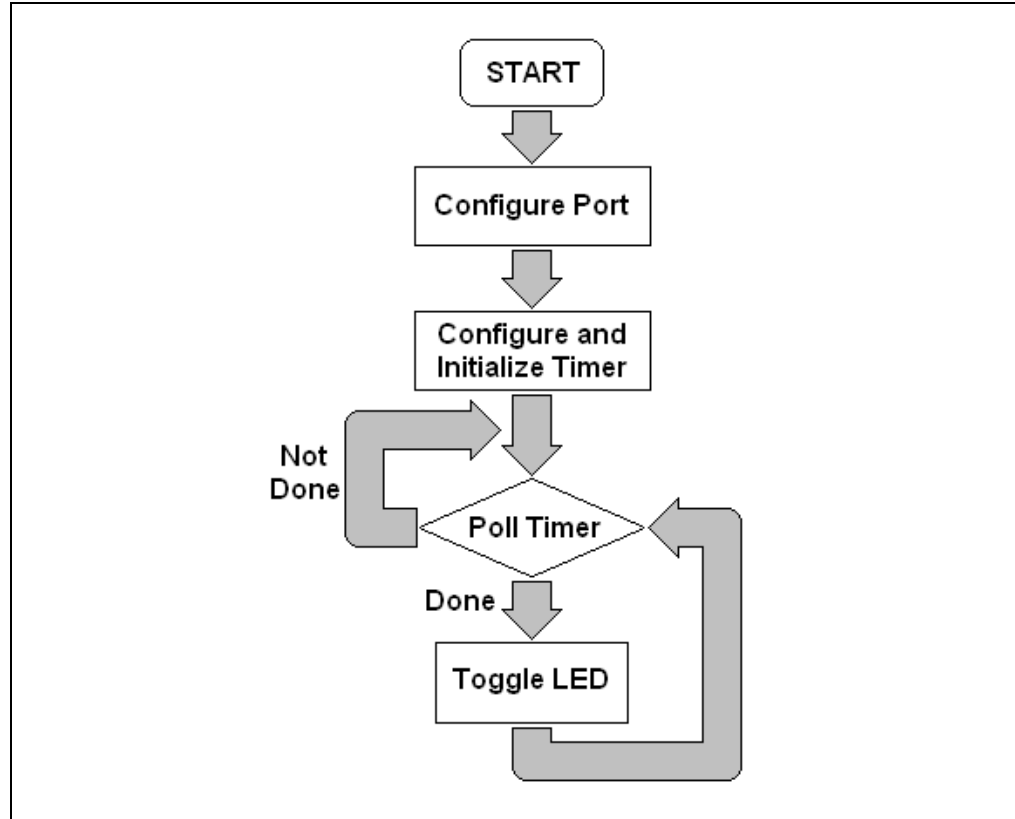
1.2.5.2 BONUS

Modify the code and setup a 32-bit `TIMER` to blink the LED at a 5 sec interval. Show your calculations for the timer delay.

1.2.5.3 PERTINENT INFORMATION

The basic program flow is illustrated in [Figure 1-3](#).

FIGURE 1-3: LAB 1 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab1-timers.s`.

LABS

NOTES:

Lab 2. Interrupts

2.1 INTERRUPTS

2.1.1 Objective

This lab covers the following concepts:

- Using interrupts on the PIC24F
- Using the MPLAB IDE Watch window to inspect real-time register values during programming debugging

2.2 PRE-LAB

2.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Family Reference Manual – Section 14. Timers (DS39704)

2.2.2 Interrupts

An interrupt (also called an exception) is an asynchronous signal or a synchronous software event intended to inform the processor that there is a need for a change in execution.

Interrupts were introduced to avoid wasting the processor's time in a polling loop. This method means the processor is fully executing while waiting for an external event, often without the ability to handle other system events.

When an interrupt occurs, it causes the processor to save its current state on the software stack and then execute a special subroutine. The Interrupt Vector Table is a list of the addresses to these special routines. The PIC24F allows for the possibility of a unique subroutine for each interrupt source.

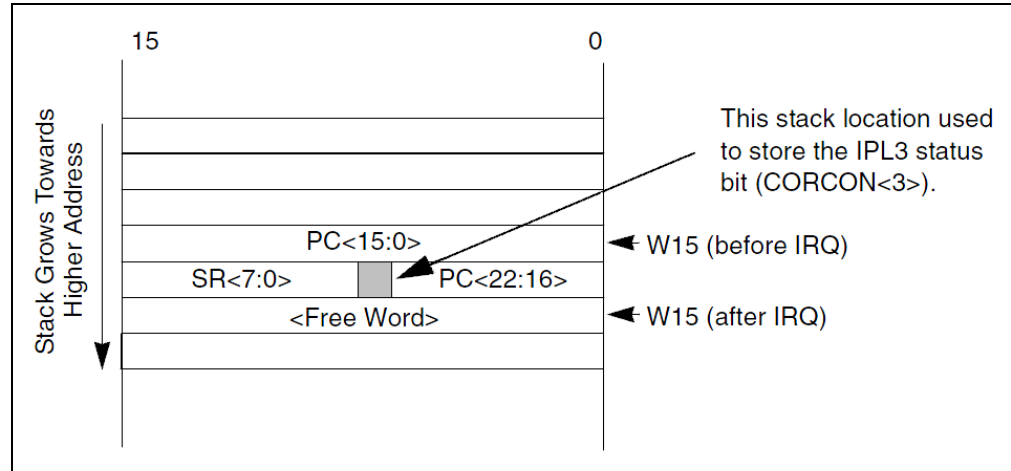
The special subroutines are known as Interrupt Service Routines (ISRs). It is good design practice for an ISR to execute as quickly as possible, so the processor should spend as little time in the ISR as possible. This becomes an issue when you have multiple, real-time, nested interrupts. In all cases, loops and polling should be avoided in any ISR.

The main difference between an ISR and a normal subroutine is that before executing an ISR, the processor saves its current state so that when the ISR is complete, the processor state can be exactly reproduced and program flow will continue as if the interrupt did not occur (with the exception of any changes caused by the ISR itself). As a result, the last instruction of an ISR must be a `RETFIE` instruction. This restores the processor state.

The processor saves the following information on the software stack as illustrated in [Figure 2-1](#):

- The current PC value
- The low byte of the processor STATUS register (SRL)
- The IPL3 Status bit (CORCON<3>)

FIGURE 2-1: LAB 2 STACK LOCATION



The `RETFIE` (Return from Interrupt) instruction will unstack the PC return address, IPL3 status bit and SRL register, to return the processor to the state and priority level prior to the interrupt sequence.

For the PIC24F, the Interrupt Vector Table (IVT) resides in program memory, starting at location 0x000004. It contains 126 vectors, consisting of 8 non-maskable trap vectors, plus up to 118 sources of interrupt. Each interrupt vector contains a 24-bit wide address. The value programmed into each interrupt vector location is the starting address of the associated Interrupt Service Routine (ISR).

The Alternate Interrupt Vector Table (AIVT) is located after the IVT, as shown in [Figure 2-2](#). Access to the AIVT is provided by the ALTIVT control bit (INTCON2<15>). If the ALTIVT bit is set, all interrupt and exception processes will use the alternate vectors instead of the default vectors. The alternate vectors are organized in the same manner as the default vectors.

FIGURE 2-2: LAB 2 PRIMARY AND ALTERNATE INTERRUPT VECTOR TABLE

Interrupt Vector Table		Alternate Interrupt Vector Table	
Reset – GOTO Instruction	0x000000	Reserved	0x000104
Reset – GOTO Address	0x000002	Reserved	
Reserved	0x000004	Reserved	
Oscillator Fail Trap Vector		Oscillator Fail Trap Vector	
Address Error Trap Vector		Address Error Trap Vector	
Stack Error Trap Vector		Stack Error Trap Vector	
Math Error Trap Vector		Math Error Trap Vector	
Reserved		Reserved	
Reserved		Reserved	
Reserved		Reserved	
Interrupt Vector 0	0x000014	Interrupt Vector 0	0x000114
Interrupt Vector 1		Interrupt Vector 1	
~		~	
~		~	
~		~	
Interrupt Vector 52	0x00007C	Interrupt Vector 52	0x00017C
Interrupt Vector 53	0x00007E	Interrupt Vector 53	0x00017E
Interrupt Vector 54	0x000080	Interrupt Vector 54	0x000180
~		~	
~		~	
~		~	
Interrupt Vector 116	0x0000FC	Interrupt Vector 116	
Interrupt Vector 117	0x0000FE	Interrupt Vector 117	0x0001FE
		Start of Code	0x000200

Decreasing Natural Order Priority

Note: A device Reset is not a true exception because the interrupt controller is not involved in the Reset process. The PIC24F device clears its registers in response to a Reset, which forces the PC to zero. The processor then begins program execution at location 0x000000.

When a UART1 Receiver interrupt occurs, the processor stores its current state on the software stack and then jumps to the address stored in the vector table for that interrupt. For the PIC24FJ256GB110, the UART1 Receiver uses interrupt 11, Interrupt Vector Table Address 0x00002A.

The processor would therefore branch to and execute the code located at the address stored at 0x00002A when a UART1 Receiver interrupt occurs.

Refer to the Microchip “PIC24FJ256GB110 Family Data Sheet” (DS39897) for a complete list of “implemented interrupt vectors”.

Setting up interrupts in assembly for the PIC24FJ256GB110 is actually quite easy. MPLAB IDE does a lot of the overhead work and each interrupt has a predefined label. [Table 2-1](#) lists some of these predefined labels.

TABLE 2-1: LAB 2 LIST OF SOME DEFINED INTERRUPTS

Label	Interrupt Source	Address
__INT0Interrupt	External Interrupt 0	0x00014
__IC1Interrupt	Input Capture 1	0x00016
__OC1Interrupt	Output Compare 1	0x00018
__T1Interrupt	TIMER1	0x0001A
__Interrupt4	Interrupt 4	0x0001C
__I2CInterrupt	Input Compare 2	0x0001E
__OC2Interrupt	Output Compare 2	0x00020
__T2Interrupt	TIMER2	0x00022
__T3Interrupt	TIMER3	0x00024
__SPI1ErrInterrupt	SPI1 Error	0x00026
__SPI1Interrupt	SPI1 Event	0x00028
__U1RXInterrupt	UART1 Receiver	0x0002A
__U1TXInterrupt	UART1 Transmitter	0x0002C
__ADC1Interrupt	ADC1 Conversion Done	0x0002E
__Interrupt14	Interrupt 14	0x00030
__Interrupt15	Interrupt 15	0x00032
__SI2C1Interrupt	I2C1 Slave Event	0x00034
__MI2C1Interrupt	I2C1 Master Event	0x00036
__CompInterrupt	Comparator Event	0x00038
__CNInterrupt	Input Change Notification	0x0003A
__INT1Interrupt	External Interrupt 0	0x0003C
__U2RxInterrupt	UART2 Receiver	000050h

To setup an interrupt in assembly, an interrupt label must first be declared. Next, the label must be used to indicate the desired ISR, not unlike any other label for a normal subroutine. The only difference between a subroutine and ISR, is that for an ISR the label is unique and it must end with a `RETFIE` instruction.

```
.global __T1Interrupt ; Declare Timer1 ISR name global
__T1Interrupt:
; Timer1 ISR code goes here
RETFIE ; Return from Interrupt Service Routine
```

To verify that the ISR label is at the correct memory location in program memory, select View>Program Memory and choose the **Machine** tab.

2.2.3 IDE Concepts Watch Window

It is often necessary to be able to determine exactly what is happening to debug a problem in the program flow. MPLAB IDE provides a Watch window for this purpose.

There are a large number of registers in the File Register window and Special Function Register (SFR) windows. However, to debug any function, only a small subset of these are required in order to determine the exact program function. To set up a specific list of registers to view, you can use the Watch window. To open the Watch window select View>Watch.

To add an item to the Watch window:

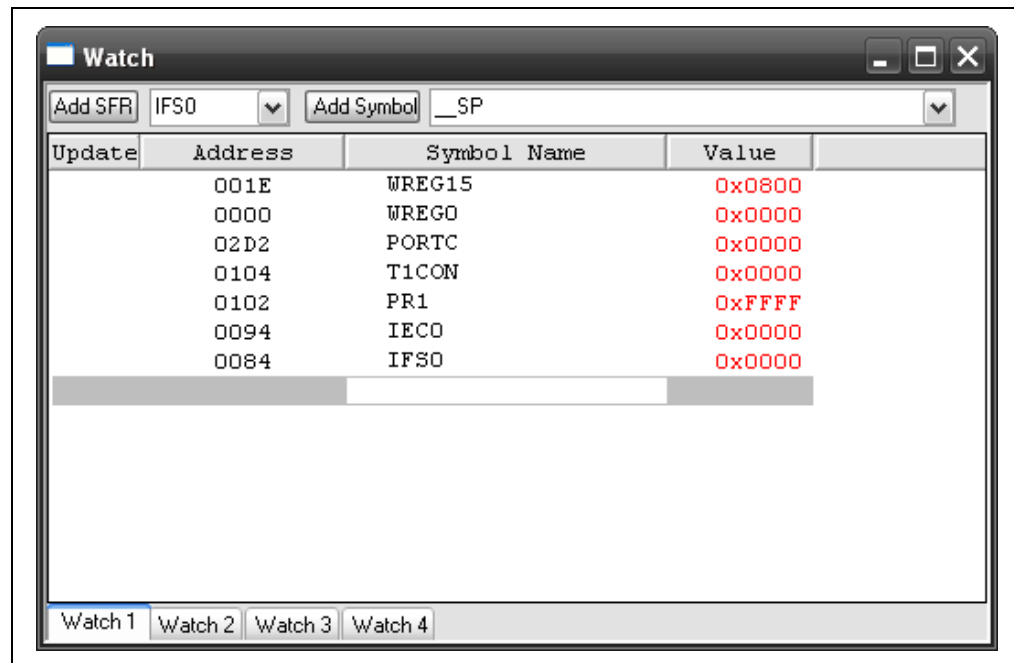
- Choose an SFR or Symbol from the drop-down list and then click the corresponding Add button
- Drag an item from the SFR, File Register or File window
- Click directly in the window under symbol name and type in the item

The file register address of the symbols is listed first, followed by the symbol name and finally the value of the symbol.

For this particular lab, the following registers may be of interest:

- WREG15 (Software Stack Pointer)
- WREG0 (Working Register)
- PORTC (LED1 is bit 2)
- T1CON (Timer1 Control Register if you are using Timer1)
- PR1 (Timer1 Preload Register)
- IEC0 (Interrupt Enable Bits)
- IFS0 (Interrupt Flag Status Bits)

FIGURE 2-3: LAB 2 REGISTERS OF INTEREST



2.3 THE LAB

2.3.1 Objective

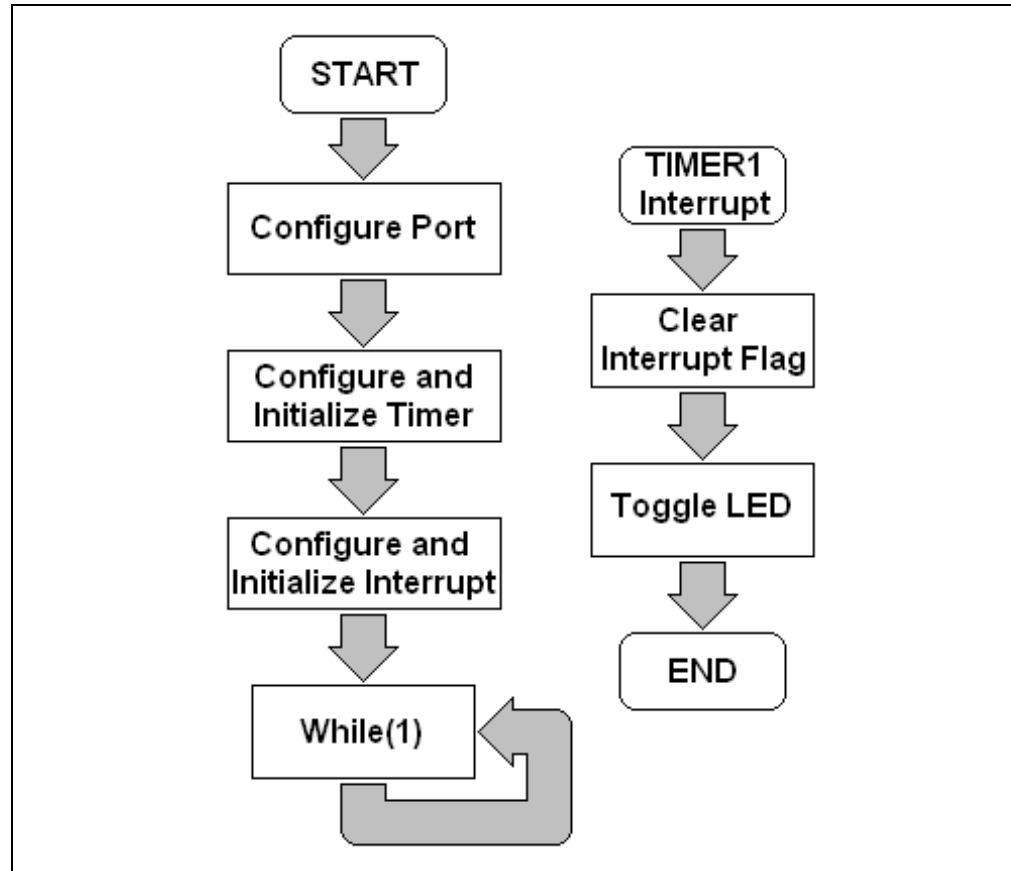
Using the Lab1 framework, program the PIC24 microcontroller to use a hardware Timer to blink an LED at a 1 second interval (LED on for 1 second, then off for 1 second). In this lab, toggle the LED inside an Interrupt Service Routine.

Use a Watch window to view register changes as you step through your code.

2.3.2 Pertinent Information

The basic program flow is depicted in [Figure 2-4](#).

FIGURE 2-4: LAB 3 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab2-interrupts.s`.

Lab 3. UART

3.1 OBJECTIVE

This lab covers the following concepts:

- PIC24F UART Hardware Module
- PIC24F Peripheral Pin Select Hardware Module
- ASCII Character Encoding

3.2 PRE-LAB

3.2.1 Reference Material

- PIC24FJ256GB110 Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Reference Manual – Section 12. Peripheral Pin Select (DS39711)
- PIC24F Reference Manual – Section 21. UART (DS39708)

3.2.2 UART (Universal Asynchronous Receiver/Transmitter)

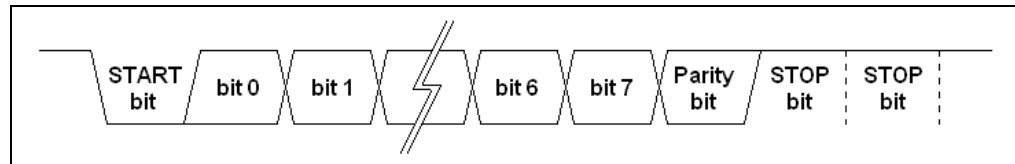
A UART is a piece of hardware that uses a shift register to convert data between serial and parallel forms. As the name implies, each UART actually contains two pieces of hardware and therefore two shift registers, a Receiver (Rx) and a Transmitter (Tx).

Simplified, a UART transmitter takes bytes (8 bits) of data and transmits them as individual bits in sequential fashion. Then another UART receiver at the other end takes those sequential bits and re-assembles them back into bytes.

The simple reason for this is that serial transmission across a single wire is much more cost effective than parallel transmission across multiple wires.

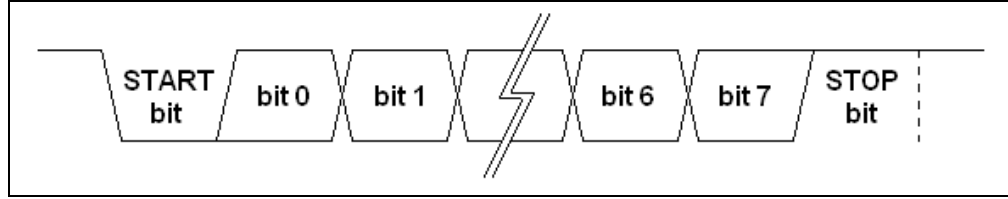
The communication is actually a little more complicated than described above. The UART is an “asynchronous” receiver/transmitter. Therefore, some synchronization between transmitter and receiver is required. To accomplish this, both the transmitter and receiver must run at the same baud rate with a START and one or two STOP bits inserted into the serial bit stream. Parity bits can also be inserted into the bit stream for error detection. [Figure 3-1](#) illustrates all possible bits in a serial bit stream.

FIGURE 3-1: LAB 3 SERIAL TRANSMISSION OF BYTE



A standard embedded UART configuration is 8 bits of data with no parity bit and one stop bit. This is also referred to as 8N1 and is illustrated in [Figure 3-2](#).

FIGURE 3-2: LAB 3 UART 8N1 SERIAL BIT STREAM



3.2.3 ASCII (American Standard Code for Information Interchange)

ASCII was based on earlier teleprinter encoding systems. It specifies the correspondence between numeric/binary values and character symbols. It was designed to allow digital devices to communicate, process and store character information.

ASCII is a seven-bit code, with the eighth bit functioning as a parity bit for error checking, if desired.

The first 32 ASCII codes (0-31 decimal) are reserved for control characters (now mostly obsolete). These codes were not intended to represent printable characters, but rather to send control information to devices. For example, character 0x0A “Line Feed” could be sent to a terminal program to skip to the next line or to a printer to advance its paper.

Control characters of interest:

- 0x08 Back space
- 0x0A Line Feed
- 0x0D Carriage Return

ASCII was designed for character symbol communication, processing and storage. It does not contain mechanisms for document layout and formatting apart from some basic line-oriented formatting. Document formatting is achieved with schemes such as markup language. HTML (Hyper Text Markup Language) is an example of one such scheme.

FIGURE 3-3: 7-BIT ASCII CODE CHART

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
00	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
10	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
20	SP	!	“	#	\$	%	&	'	()	*	+	,	-	.	/
30	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
40	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
50	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
60	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
70	p	q	r	s	t	u	v	w	x	y	Z	{		}	~	DEL
80	Ç	ü	é	â	ä	à	â	ç	ê	è	è	ï	î	ï	Ä	Å
90	É	æ	Æ	ô	ö	ö	û	ü	ÿ	Ö	Ü	ø	£	¥	Pts	f
A0	á	í	ó	ú	ñ	Ñ	ª	º	¿	¬	¬	½	¼	¾	«	»
B0	☘	☙	☚	☛	☜	☝	☞	☟	☠	☡	☢	☣	☤	☥	☦	☧
C0	ℒ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ
D0	ℒ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ	ℓ
E0	α	β	Γ	π	Σ	σ	μ	τ	Φ	Θ	Ω	δ	∞	φ	ε	η
F0	≡	±	≥	≤			+	≈	°	.	.	√	n	²	■	

Hint: To view any of the ASCII display characters, simply hold the Alt key and type the decimal number in most Windows text editors.

3.2.4 I/O Ports with Peripheral Pin Select (PPS)

The Peripheral Pin Select (PPS) module controls the connection between digital hardware modules and the external device pins. Traditionally, microcontroller pins would only be internally connected to a specific hardware module with a dedicated function. As microcontrollers became more complex, configuration options were provided to select the I/O of a hardware module from a small number of preconfigured options.

The PPS module allows for ultimate configuration. For any pin controlled by the PPS module, a completely flexible connection matrix exists between the I/O of the hardware module and the device pins. Thus, any hardware module can be connected to any device pin. This allows for a much more flexible user configuration of hardware modules.

The hardware modules managed by the PPS are all digital-only peripherals. These include general serial communications (UART and SPI), general purpose timer clock inputs, timer related peripherals (input capture and output compare) and external interrupt inputs.

In comparison, some digital-only peripheral modules are not currently included in the Peripheral Pin Select feature. This is because the peripheral's function requires special I/O circuitry on a specific port and cannot be easily connected to multiple pins. These modules include I²C™, high-speed communication (Ethernet and USB), change notification inputs, RTCC alarm output and all modules with analog inputs, such as the A/D Converter.

A key difference between remappable and non-remappable peripherals is that remappable peripherals are not associated with a default I/O pin. The peripheral must always be assigned to a specific I/O pin before it can be used. In contrast, non-remappable peripherals are always available on a default pin, assuming that the peripheral is active and not conflicting with another peripheral.

When a remappable peripheral is active on a given I/O pin, it takes priority over all other digital I/O and digital communication peripherals associated with the pin. Priority is given regardless of the type of peripheral that is mapped. Remappable peripherals never take priority over any analog functions associated with the pin.

The PIC24FJ256GB110 has a total of 44 remappable pins, which includes 12 input only pins:

- RP0-RP31 Remappable Peripheral (input or output)
- RPI32-RPI43 Remappable Peripheral (input ONLY)

3.2.5 PPS Input Mapping

The PIC24FJ256GB110 has 21 Input Remappable Peripheral Registers (RPINRx).

TABLE 3-1: LAB 3 INPUT REMAPPABLE PERIPHERALS REGISTERS

Register	Input Name	Function Mapping Bits
RPINR0	External Interrupt 1 (INT1)	INT1R<5:0>
RPINR1	External Interrupt 3 (INT3) External Interrupt 2 (INT2)	INT3R<5:0> INT2R<5:0>
RPINR2	External Interrupt 4 (INT4)	INT4R<5:0>
RPINR3	Timer3 External Clock (T3CK) Timer2 External Clock (T2CK)	T3CKR<5:0> T2CKR<5:0>
RPINR4	Timer5 External Clock (T5CK) Timer4 External Clock (T4CK)	T5CKR<5:0> T4CKR<5:0>
RPINR7	Input Capture 2 (IC2) Input Capture 1 (IC1)	IC2R<5:0> IC1R<5:0>
RPINR8	Input Capture 4 (IC4) Input Capture 3 (IC3)	IC4R<5:0> IC3R<5:0>
RPINR9	Input Capture 6 (IC6) Input Capture 5 (IC5)	IC6R<5:0> IC5R<5:0>
RPINR10	Input Capture 8 (IC8) Input Capture 7 (IC7)	IC8R<5:0> IC7R<5:0>
RPINR11	Output Compare Fault B (OCFB) Output Compare Fault A (OCFA)	OCFBR<5:0> OCFAR<5:0>
RPINR15	Input Capture 9 (IC9)	IC9R<5:0>
RPINR17	UART3 Receive (U3RX)	U3RXR<5:0>
RPINR18	UART1 Clear to Send (U1CTS) UART1 Receive (U1RX)	U1CTSR<5:0> U1RXR<5:0>
RPINR19	UART2 Clear to Send (U2CTS) UART2 Receive (U2RX)	U2CTSR<5:0> U2RXR<5:0>
RPINR20	SPI1 Clock Input (SCK1IN) SPI1 Data Input (SDI1)	SCK1R<5:0> SDI1R<5:0>
RPINR21	UART3 Clear to Send (U3CTS) SPI1 Slave Select Input (SS1IN)	U3CTSR<5:0> SS1R<5:0>
RPINR22	SPI2 Clock Input (SCK2IN) SPI2 Data Input (SDI2)	SCK2R<5:0> SDI2R<5:0>
RPINR23	SPI2 Slave Select Input (SS2IN)	SS2R<5:0>
RPINR27	UART4 Clear to Send (U4CTS) UART4 Receive (U4RX)	U4CTSR<5:0> U4RXR<5:0>
RPINR28	SPI3 Clock Input (SCK3IN) SPI3 Data Input (SDI3)	SCK3R<5:0> SDI3R<5:0>
RPINR29	SPI3 Slave Select Input (SS3IN)	SS3R<5:0>

At first glance, this appears confusing, but it is actually a simple mapping once understood. Since we are mapping inputs, we can use any of the RP0-RP31 or RPI32-RPI43 pins. Start by picking the desired peripheral pin and then write the PR or PRI number to that register.

For example:

Mapping	Action
Map "UART1 Receiver" to RP30	Write the RP value 30 decimal = 0x1E hex into U3RXR<5:0>
Map "SPI3 Data Input" to RPI38	Write the RPI value 38 decimal = 0x26 hex into SDI3R<5:0>

3.2.6 PPS Output Mapping

The PIC24FJ256GB110 has 16 Output Remappable Peripheral Registers (RPORx).

The output mapping is the exact opposite to the input mapping. For input mapping, the port number is written to a function register. For output mapping, the function number is written to the port register. This makes sense since each output pin can only be connected to one peripheral output.

TABLE 3-2: LAB 3 PPS OUTPUT CONFIGURATION

Output Function Number	Function	Output Name
0	NULL	Null
1	C1OUT	Comparator 1 Output
2	C2OUT	Comparator 2 Output
3	U1TX	UART1 Transmit
4	U1RTS	UART1 Request To Send
5	U2TX	UART2 Transmit
6	U2RTS	UART2 Request To Send
7	SDO1	SPI1 Data Output
8	SCK1OUT	SPI1 Clock Output
9	SS1OUT	SPI1 Slave Select Output
10	SDO2	SPI2 Data Output
11	SCK2OUT	SPI2 Clock Output
12	SS2OUT	SPI2 Slave Select Output
13-17	Unused	N/C
18	OC1	Output Compare 1
19	OC2	Output Compare 2
20	OC3	Output Compare 3
21	OC4	Output Compare 4
22	OC5	Output Compare 5
23	OC6	Output Compare 6
24	OC7	Output Compare 7
25	OC8	Output Compare 8
26-27	Unused	N/C
28	OC1	UART3 Transmit
29	U3RTS	UART3 Request To Send
30	U4TX	UART4 Transmit
31	U4RTS	UART4 Request To Send
32	SDO3	SPI3 Data Output
33	SCK3OUT	SPI3 Clock Output
34	SS3OUT	SPI3 Slave Select Output
35	OC9	Output Compare 9
36	C3OUT	Comparator 3 Output
37-63	Unused	N/C

Setting the RPORx register with the listed value assigns that output function to the associated RPn pin.

Each RPORx register holds function numbers for two PR port pins.

- RPOR0<13:8> = RP1R<5:0> = RP1 Output Pin Mapping bits
- RPOR0<5:0> = RP0R<5:0> = RP0 Output Pin Mapping bits

For example:

Mapping	Action
Map UART 3 Tx to PR31 (pin 39)	Write the value 28 decimal = 0x1C hex to RP31R<5:0>
Map the Comparator 2 Output to RP9 (pin 33)	Write the value 2 decimal = 0x02 hex to RP9R<5:0>

3.2.7 PPS Register Lock/Unlock Sequence

Under normal operation, writes to the RPINRx and RPORx registers are not allowed.

Attempted writes will appear to execute normally, but the contents of the registers will remain unchanged. To change these registers, they must be unlocked in hardware. The register lock is controlled by the IOLOCK bit (OSCCON<6>). Setting IOLOCK prevents writes to the control registers; clearing IOLOCK allows writes.

To set or clear IOLOCK, a specific command sequence must be executed:

1. Write 46h to OSCCON<7:0>.
2. Write 57h to OSCCON<7:0>.
3. Clear (or set) IOLOCK as a single operation.

Register unlock sequence in assembly:

```
MOV    #OSCCON, W1
MOV    #0x46, W2
MOV    #0x57, W3
MOV.B  #W2, [W1]           ; Must be a MOV.B
MOV.B  #W3, [W1]           ; Must be a MOV.B
BCLR   #OSCCON, #IOLOCK   ; Unlock Registers
```

Register lock sequence in assembly:

```
MOV    #OSCCON, W1
MOV    #0x46, W2
MOV    #0x57, W3
MOV.B  #W2, [W1]           ; Must be a MOV.B
MOV.B  #W3, [W1]           ; Must be a MOV.B
BSET   #OSCCON, #IOLOCK   ; Lock Registers
```

Notes: The RPINRx and RPORx registers are unlocked after reset. Therefore, you will be able to write these registers without having to perform the register unlock sequence.

In real world applications, it is important to protect the PPS configuration. Accidental changes to the RPINRx and RPORx registers during normal operation would be extremely undesirable. Under normal circumstances the code would configure the RPINRx and RPORx registers and then immediately run the register lock sequence.

It should also be noted that if the configuration bit IOL1WAY bit (CW2<4>) is set, when once the IOLOCK bit is set once, it can not be cleared again (not even with the unlock sequence) and the PPS registers cannot be written again.

3.3 THE LAB

3.3.1 Objective

Program the PIC24 microcontroller to use the UART to communicate with a program such as Windows HyperTerminal. Setup the UART for 8N1 with 9600 baud, and show your baud rate calculations.

Use the UART receiver interrupt to inform you when a character is available. Process the character received and transmit back the same character but in the opposite case, i.e. if a lower case “e” is received, then “E” would be transmitted back.

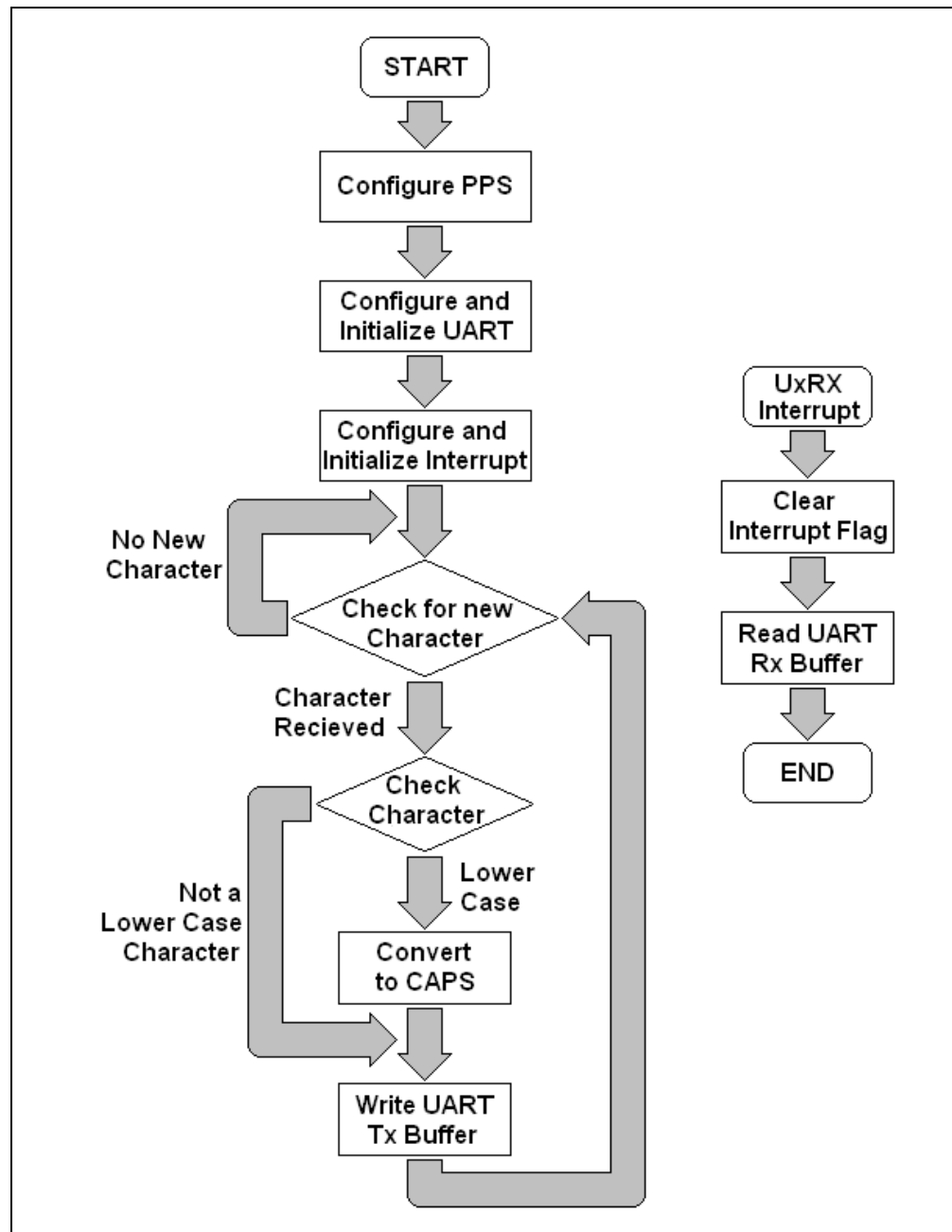
Connect the RS232 port of the MX Educational Target to a PC and use a terminal emulator (e.g. HyperTerminal that comes with Windows) to communicate with the embedded processor using ASCII characters.

3.3.2 Pertinent Information

The PIC24FJ256GB110 has 4 configurable UARTS. This lab will only use one.

Configure PPS for the chosen UART peripheral. To use the DB9 RS-232 port “CH1” on the MX Educational Target board, map Rx to RP30 (pin 52) and Tx to RP15 (pin 53). To enable handshaking (not required), map RTS to RP5 (pin 48) and CTS to RPI43 (pin 47). The basic program flow is depicted in [Figure 3-4](#).

FIGURE 3-4: LAB 3 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab3-uart.s`.

Lab 4. Watchdog Timer (WDT)

4.1 OBJECTIVE

This lab covers the following concepts:

- PIC24F Watchdog Module
- PIC24F Configuration Bits

4.2 PRE-LAB

4.2.1 Reference Material

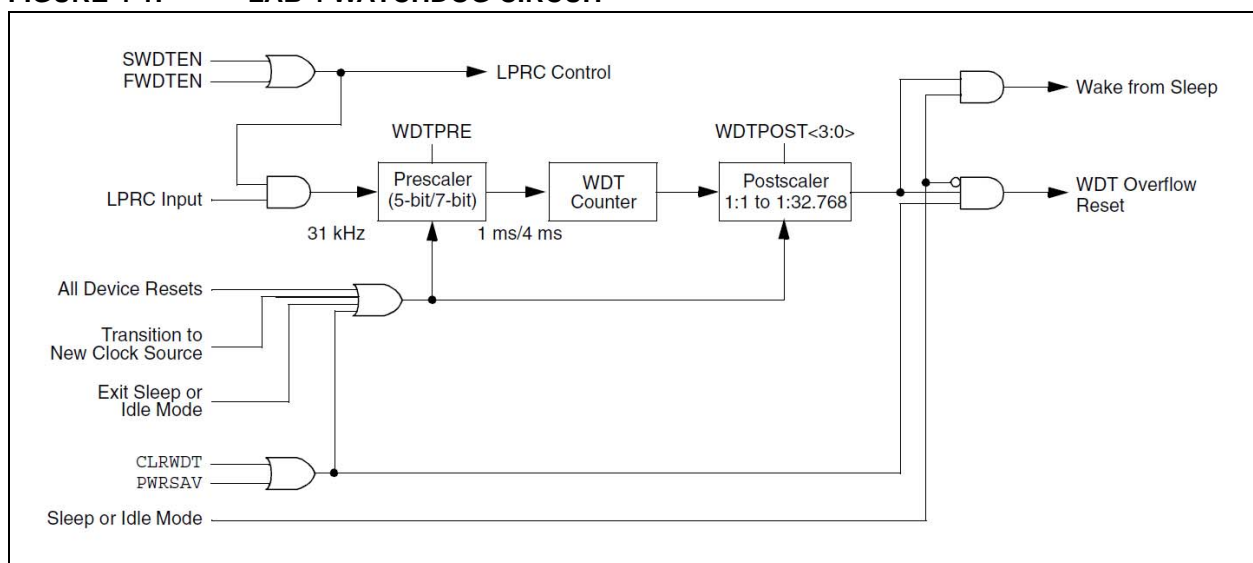
- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Family Reference Manual – Section 9. Watchdog Timer (DS39697)
- PIC24F Family Reference Manual – Section 12. Peripheral Pin Select (DS39711)
- PIC24F Family Reference Manual – Section 14. Timers (DS39704)
- PIC24F Family Reference Manual – Section 21. UART (DS39708)

4.2.2 The Watchdog Timer

The primary function of the Watchdog Timer (WDT) is to reset the microcontroller in the event of a software malfunction. It can also be used to wake the device from Sleep or Idle mode.

The WDT is a free-running timer which uses the low-power RC oscillator. It requires no external components so the WDT will continue to operate even if the system's primary clock source (e.g., the crystal oscillator) is stopped.

FIGURE 4-1: LAB 4 WATCHDOG CIRCUIT



When enabled, the WDT will increment until it overflows or “times out”. A system should be designed so that under normal circumstances the WDT will never “time out”. Thus, if the situation occurs, then something is wrong and drastic action should be taken. The typical reason for this to happen is due to a firmware bug that gets stuck in an infinite loop.

If the WDT “times out”, it will force a device reset. There are few options to deal with errors like this in an embedded system. The only real option is to reset the device and restart code execution. If the error condition is resolved the system will restart successfully. If the error condition remains, the system may be remain in a loop of continuously resetting.

In order to prevent the WDT reset, the application must periodically clear the WDT using the instruction CLRWDT. This resets the timer to ‘0’. The application would typically do this once through the main control loop.

The exception to this is during Sleep or Idle modes. In these cases, the processor will continue execution from the point where the `PWRSVAV` instruction was executed. Thus, the WDT can be an integral part of a low-power design.

In either case, it is possible to determine if the processor has experienced a WDT time out. The `WDTO` bit (`RCON<4>`) will be set to indicate that the device reset or wake-up event was due to a WDT time-out. If the WDT wakes the CPU from Sleep or Idle mode, the `SLEEP` status bit (`RCON<3>`) or `IDLE` status bit (`RCON<2>`) will also be set to indicate that the device was previously in a power-saving mode.

4.2.3 Watchdog Timer Period

WDT Period (ms) = Prescaler Factor x Postscaler Factor

Prescaler Factor:

1 for `WDTPRE` is ‘0’

4 for `WDTPRE` is ‘1’

Postscaler Factor:

1/Postscaler Ratio

TABLE 4-1: LAB 4 WATCHDOG PERIODS

Postscaler Setting (<code>WDTPS3:WDTPS0</code>)	Postscaler Ratio (1/Postscaler Factor)	Timeout Period	
		5-Bit Prescaler (<code>FWPSA = 0</code>)	7-Bit Prescaler (<code>FWPSA = 1</code>)
0000	1:1	1 ms	4 ms
0001	1:2	2 ms	8 ms
0010	1:4	4 ms	16 ms
0011	1:8	8 ms	32 ms
0100	1:16	16 ms	64 ms
0101	1:32	32 ms	128 ms
0110	1:64	64 ms	256 ms
0111	1:128	128 ms	512 ms
1000	1:256	256 ms	1.024s
1001	1:512	512 ms	2.048s
1010	1:1024	1.024s	4.096s
1011	1:2048	2.048s	8.192s
1100	1:4096	4.096s	16.384s
1101	1:8192	8.192s	32.768s
1110	1:16384	16.384s	65.536s
1111	1:32768	32.768s	131.072s

4.2.4 Configuration Bits

In general, most device configuration is done at run time by executing various instructions. However, there are some lower level functions that are only configurable at programming time. The WDT pre-scaler and post-scaler are two of the configuration settings that are only configurable at programming time.

The top three words of on-chip program memory in the PIC24FJ256GB110 family of devices are reserved for configuration information. On device Reset, the configuration information is copied into the appropriate run-time registers.

The Configuration bits are mapped starting at program memory location F80000h. Note that this address is beyond the user program memory space. In fact, it belongs to the configuration memory space (800000h-FFFFFFh), which can only be accessed using table reads and table writes and cannot be modified during normal code execution.

Each lab has a file named `config_bits_pic24fj256gb110.inc` that includes the values used for every Configuration bit available in this family of devices. It also includes comments that describe each possible setting for each field.

4.3 THE LAB

4.3.1 Objective

Modify the file `config_bits_pic24fj256gb110.inc` to change the WDT period and write code to monitor the Watchdog Timer. Use the controller itself to measure the Watchdog Timer period and compare to the expected values.

Since the WDT time out will cause the processor to reset, a clever scheme is needed to measure the WDT time out. One way to do this is to configure a timer for a 1 ms period. Each time the timer expires, increment a local variable and send out the value of that variable through the UART to be monitored using a PC terminal emulator (like HyperTerminal).

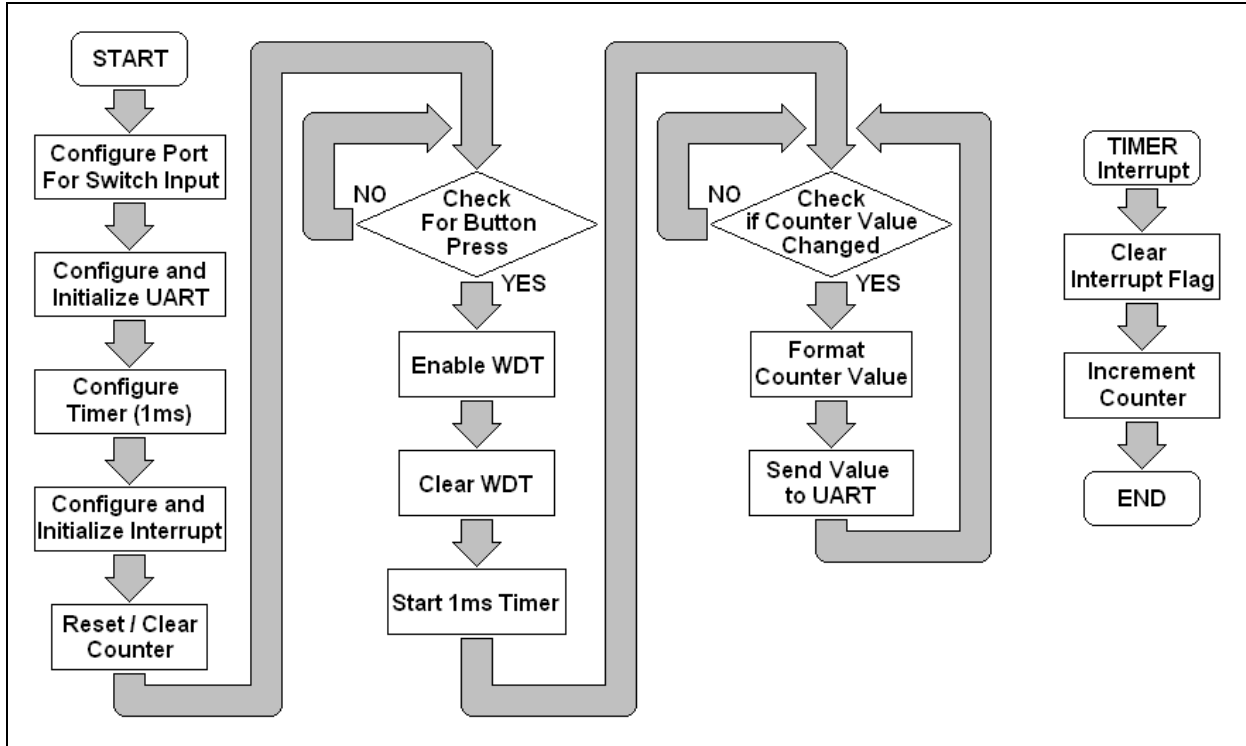
When the Watchdog Timer expires, it will reset the controller and thus the value of the timer will be reset to zero. By waiting for a button press on startup, it is possible to keep the controller from continuously overwriting the previous values in the terminal emulator on power up. In this case, the last counter value that was written to the UART should be the WDT period.

The UART needs to run at 115200 baud. At that baud rate, a character will take about 87 us to be transmitted. This means there will be just enough time for some processing and to transmit about 10 characters between the 1 ms interrupts. If more characters are sent out the UART, then the resolution of the timer must increase. This will decrease the accuracy of the measurement, but may allow for a more informative message. This is a typical design trade-off that happens in real embedded system design.

4.3.2 Pertinent Information

The basic program flow is depicted in [Figure 4-2](#).

FIGURE 4-2: LAB 4 BASIC PROGRAM FLOW



The application will have to enable the WDT in code with the instruction `BSET RCON, #SWDTEN`.

The WDT, prescaler and postscaler are reset by a `CLRWDT` instruction during normal code execution.

Remember that when the WDT “overflows”, it will reset the microcontroller. This means all volatile RAM values will be reset to zero.

The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab4-wdt.s`.

Lab 5. Comparator

5.1 OBJECTIVE

This lab covers the following concepts:

- PIC24F Comparator Hardware module
- PIC24F Comparator Voltage Reference Module

5.2 PRE-LAB

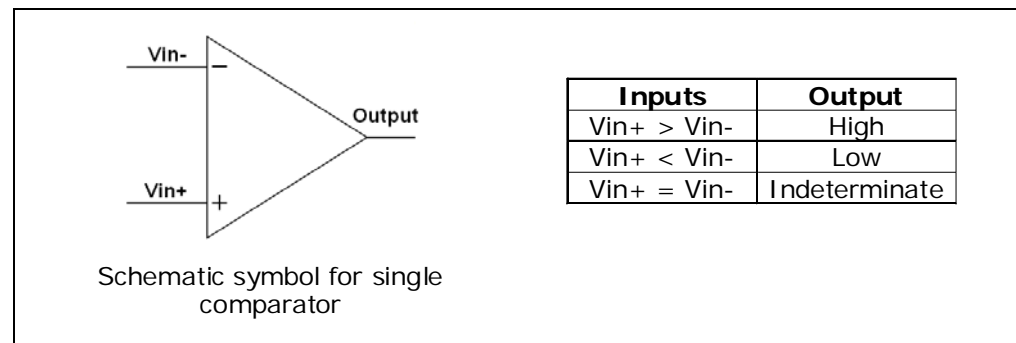
5.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Family Reference Manual – Section 12. Peripheral Pin Select (DS39711)
- PIC24F Family Reference Manual – Section 19. Dual Comparator Module (DS39710)
- PIC24F Family Reference Manual – Section 20. Comparator Voltage Reference Module (DS39708)

5.2.2 Comparator

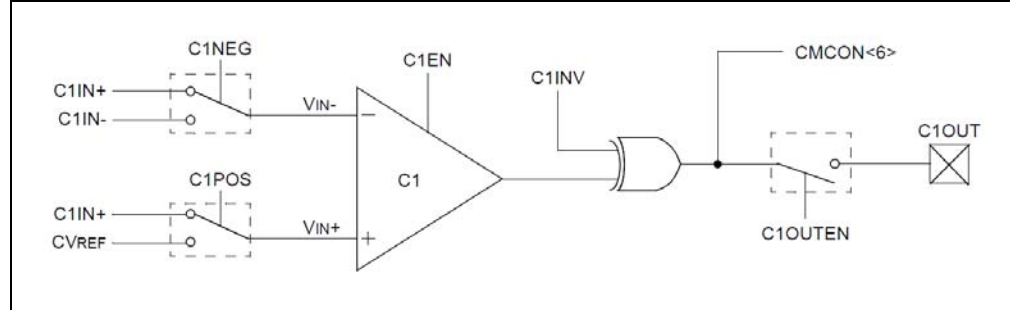
A comparator compares two analog inputs and switches its output to indicate which is larger. When the V_{in+} input is larger than to the V_{in-} input, the output is high. When the V_{in-} input is larger than the V_{in+} input, the output is low.

FIGURE 5-1: LAB 5 COMPARATOR DESCRIPTION



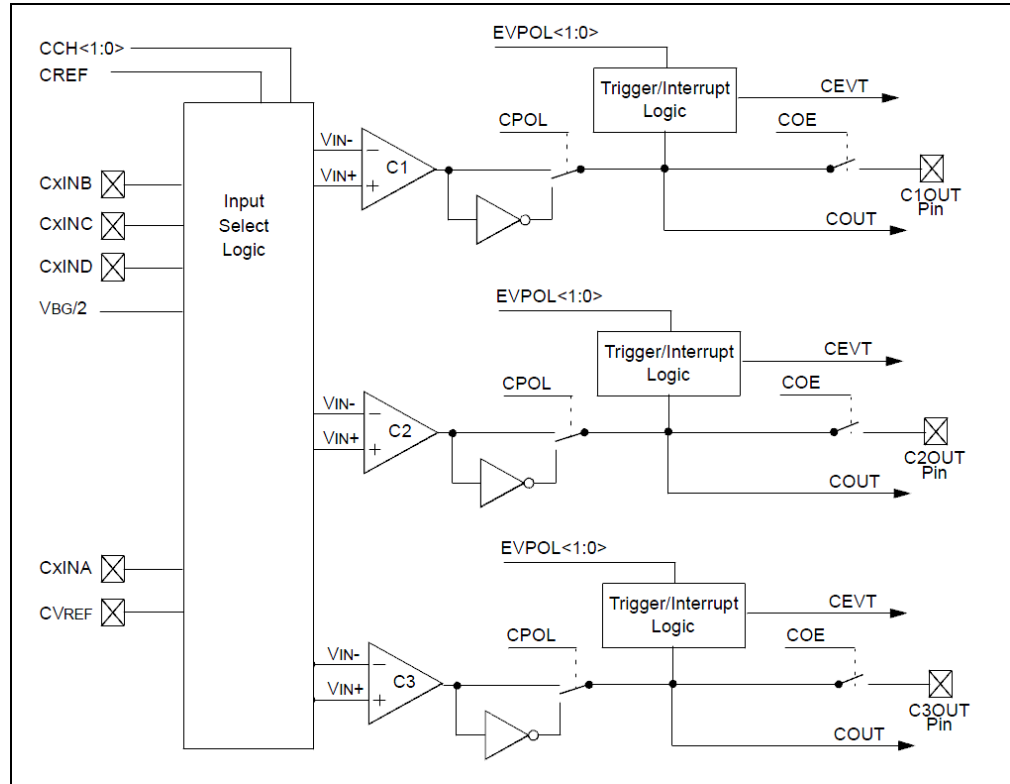
The PIC24FJ256GB110 comparator input/output configurations are completely user programmable using the CMxCON registers.

FIGURE 5-2: LAB 5 COMPARATOR I/O CONFIGURATION



The PIC24FJ256GB110 contains three analog comparators.

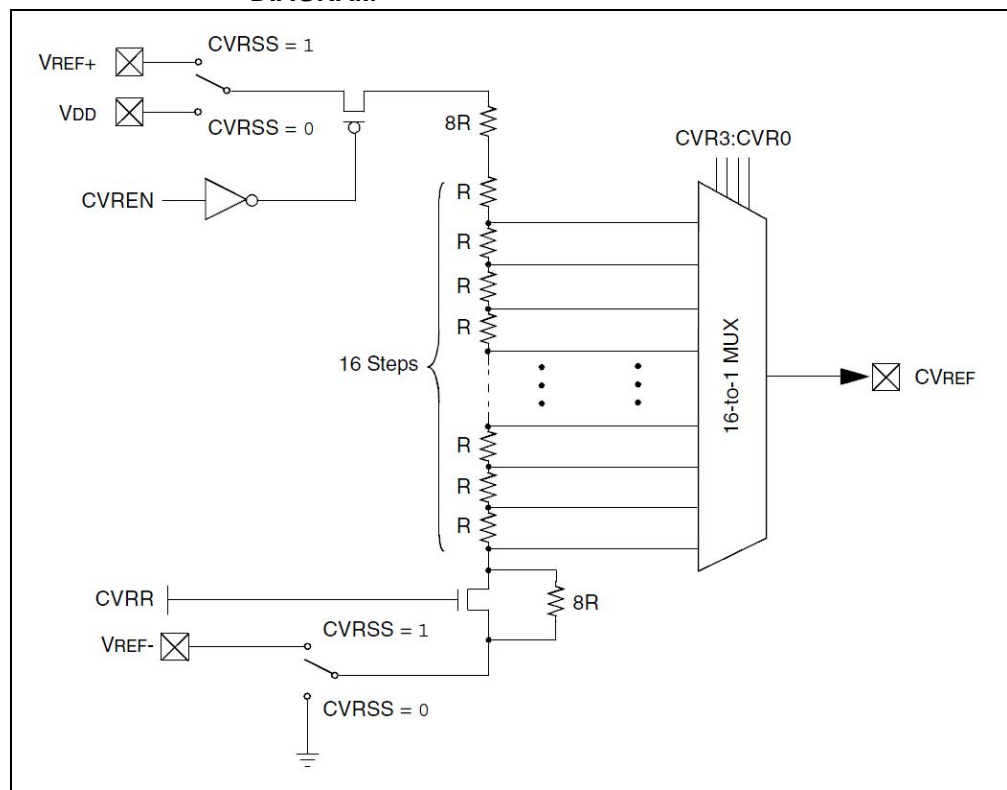
FIGURE 5-3: LAB 5 TRIPLE COMPARATOR BLOCK DIAGRAM



5.2.3 Voltage Reference Module

The PIC24FJ256GB110 has a built-in voltage reference. The voltage reference is a 16-tap, resistor ladder network that provides a selectable reference voltage. Although its primary purpose is to provide a reference for the analog comparators, it may also be used independently of them.

FIGURE 5-4: LAB 5 COMPARATOR VOLTAGE REFERENCE BLOCK DIAGRAM



The voltage reference is configurable through the CVRCON register.

TABLE 5-1: REFERENCE VOLTAGE WITH VDD = 3.3V

CVR3:CVR0	CVRR = 0		CVRR = 1	
	CVREF	CVREF (VDD 3.3V)	CVREF	CVREF (VDD 3.3V)
0000	VDD x 8/32	0.83V	VDD x 0/24	0V
0001	VDD x 9/32	0.93V	VDD x 1/24	0.14V
0010	VDD x 10/32	1.03V	VDD x 2/24	0.28V
0011	VDD x 11/32	1.13V	VDD x 3/24	0.41V
0100	VDD x 12/32	1.24V	VDD x 4/24	0.55V
0101	VDD x 13/32	1.34V	VDD x 5/24	0.69V
0110	VDD x 14/32	1.44V	VDD x 6/24	0.83V
0111	VDD x 15/32	1.55V	VDD x 7/24	0.96V
1000	VDD x 16/32	1.65V	VDD x 8/24	1.10V
1001	VDD x 17/32	1.75V	VDD x 9/24	1.24V
1010	VDD x 18/32	1.86V	VDD x 10/24	1.38V
1011	VDD x 19/32	1.96V	VDD x 11/24	1.51V
1100	VDD x 20/32	2.06V	VDD x 12/24	1.65V
1101	VDD x 21/32	2.17V	VDD x 13/24	1.79V
1110	VDD x 22/32	2.27V	VDD x 14/24	1.93V
1111	VDD x 23/32	2.37V	VDD x 15/24	2.06V

If CVRR = 1: Voltage Reference = ((CVR3:CVR0)/24) x (CVRSRC)

If CVRR = 0: Voltage Reference = (CVRSRC/4) + ((CVR3:CVR0)/32) x (CVRSRC)

5.3 THE LAB

5.3.1 Objective

Set up the Comparator and Voltage Reference module to create a simple voltage meter. Use the breadboard area on the MX Educational Target board to provide a very simple adjustable voltage source.

Setup Comparator 1 with $V_{in-} = C1INB$ (pin 21) and with $V_{in+} = CVref$. Adjust the voltage reference to measure the input voltage on C1INB. Format the results and send back the results as a three-digit number formatted similar to "1.65V".

To make things a little easier, simply step through the reference voltages with $CVRR = 1$ (0V to 2.06V), monitor when the comparator output changes and report that voltage back through the serial port.

To make debugging easier, configure the Voltage Reference module to output the voltage level on the CVREF pin ($CVRCON<7> = CVREN$). This will enable you to view the behavior of the VRef voltage with an external multi-meter.

Note: Make sure to wait for the voltage reference to stabilize before checking the output of the comparator.

Verify the measured results with a multi-meter.

5.3.2 Bonus

Modify the code to use a divide and conquer method of determining the voltage with a minimum number of iterations. Also use all reasonable reference voltages available, ie. $CVRR = 0$ and $CVRR = 1$ for the full range from 0V to 2.37V with more steps.

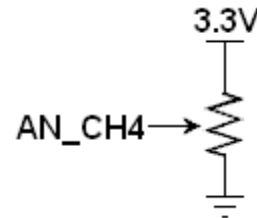
5.3.3 Pertinent Information

You will need to setup a simple external variable voltage circuit to vary an external voltage signal. This will be used as the external voltage to drive comparators V_{in-} input.

Wire a potentiometer as illustrated below.



Potentiometer

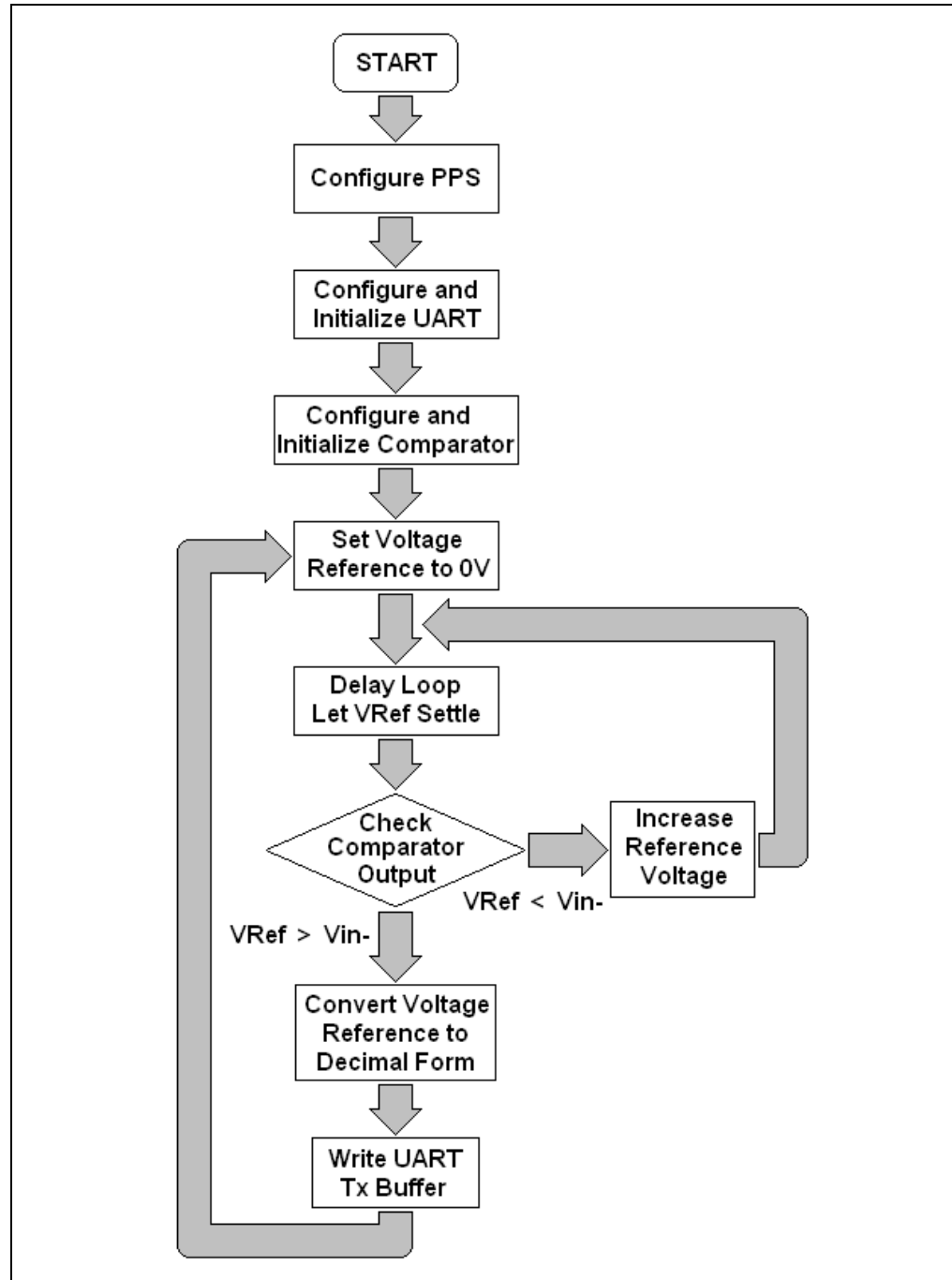


Potentiometer Circuit

The potentiometer (also known as a "pot") is a three terminal resistor that will form an adjustable voltage divider. This will allow adjustment of the voltage at the comparators V_{in-} input between 0 and 3.3V.

The basic program flow is depicted in [Figure 5-5](#).

FIGURE 5-5: LAB 5 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab5-comparator.s`.

LABS

NOTES:

Lab 6. Analog-to-Digital Converter

6.1 OBJECTIVE

Configure the PIC24FJ256GB110 10-bit A/D converter module to create a simple voltage meter.

6.2 PRE-LAB

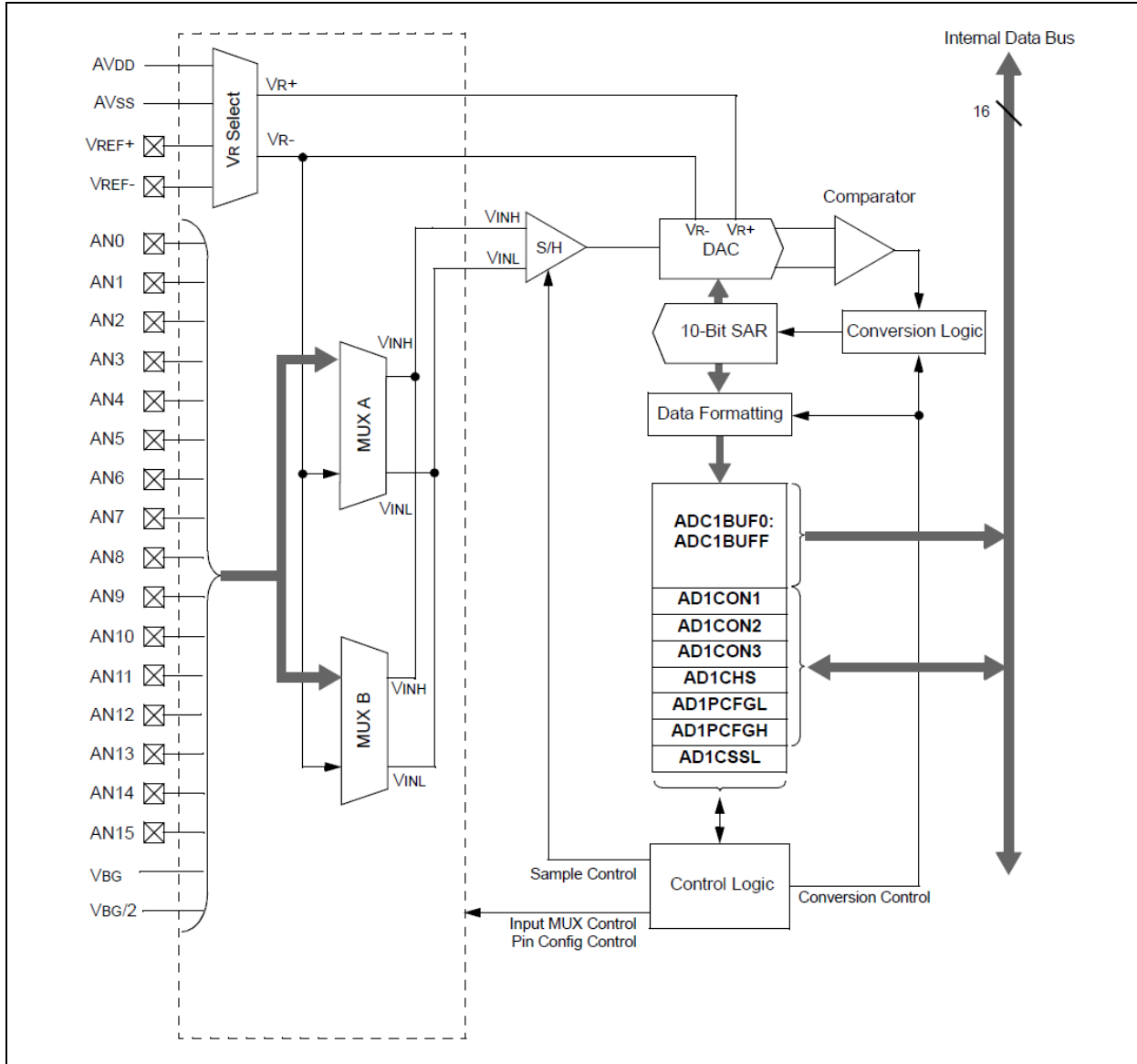
6.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Family Reference Manual – Section 17. 10-Bit A/D Converter (DS39705)

6.2.2 A/D Converter

The PIC24FJ256GB110 incorporates a 10-bit successive approximation converter. [Figure 6-1](#) illustrates the internal structure of the A/D Converter module.

FIGURE 6-1: ADC MODULE DIAGRAM



Each ADC module has the following registers that control the behavior of the module. For example, ADC1 has the following registers:

- AD1CON1: A/D Control Register 1
- AD1CON2: A/D Control Register 2
- AD1CON3: A/D Control Register 3
- AD1CHS: A/D Input Select Register
- AD1PCFGL: A/D Port Configuration Register (Low)
- AD1PCFGH: A/D Port Configuration Register (High)
- AD1CSSL: A/D Input Scan Select Register

To configure the A/D module, the following steps must be executed:

1. Configure port pins as analog inputs and/or select band gap reference inputs (AD1PCFGL<15:0> and AD1PCFGH<1:0>)
2. Select voltage reference source to match expected range on analog inputs (AD1CON2<15:13>)
3. Select the analog conversion clock to match desired data rate with processor clock (AD1CON3<7:0>)
4. Select the appropriate sample/conversion sequence (AD1CON1<7:5> and AD1CON3<12:8>)
5. Select how conversion results are presented in the buffer (AD1CON1<9:8>)
6. Select interrupt rate (AD1CON2<5:2>)
7. Turn on A/D module (AD1CON1<15>)

If interrupts are used, then the following steps must be executed during configuration of the A/D module:

1. Clear the AD1IF bit
2. Select A/D interrupt priority
3. Enable the interrupt; set AD1IE bit

6.2.3 Conversion Time

The A/D Converter has a maximum rate at which conversions may be completed. An analog module clock, T_{AD} , controls the conversion timing. The A/D conversion requires 12 clock periods ($12 T_{AD}$). The A/D clock is derived from the device instruction clock.

The period of the A/D conversion clock is software selected using a 6-bit counter. There are 64 possible options for T_{AD} , specified by the ADCS bits in the AD1CON3 register.

For correct A/D conversions, the A/D conversion clock (T_{AD}) must be selected to ensure a minimum T_{AD} time of 75 ns.

$$F_{OSC} = 32 \text{ MHz}$$

$$T_{CY} = 2/F_{OSC} = 2/32 \text{ MHz} = 62.5 \text{ ns}$$

$$ADCS\langle 7:0 \rangle = (T_{AD}/T_{CY}) - 1$$

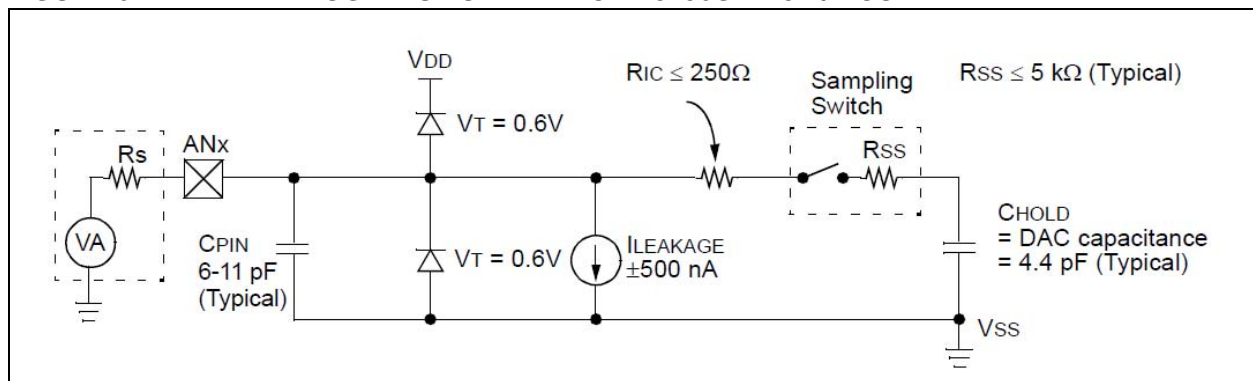
$$T_{AD} = T_{CY} \cdot (ADCS\langle 7:0 \rangle + 1)$$

For $T_{AD} > 75 \text{ ns}$ we will require ADCS of at least 1.

6.2.4 Sample Time

Figure 6-2 the analog input model of the 10-bit A/D Converter of the PIC24FJ256GB110.

FIGURE 6-2: ANALOG INPUT OF THE PIC24FJ256GB110 A/D CONVERTER



The total sampling time for the A/D is a function of the holding capacitor charge time.

For the A/D Converter to meet its specified accuracy, the charge holding capacitor (CHOLD) must be allowed to fully charge to the voltage level on the analog input pin. The source impedance (RS), the interconnect impedance (RIC) and the internal sampling switch (RSS) impedance combine to directly affect the time required to charge CHOLD. The combined impedance of the analog sources must, therefore, be small enough to fully charge the holding capacitor within the chosen sample time. To minimize the effects of pin leakage currents on the accuracy of the A/D Converter, the maximum recommended source impedance, RS, is 2.5 k. After the analog input channel is selected (changed), this sampling function must be completed prior to starting the conversion. The internal holding capacitor will be in a discharged state prior to each sample operation.

At least 1 TAD time period should be allowed between conversions for the sample time.

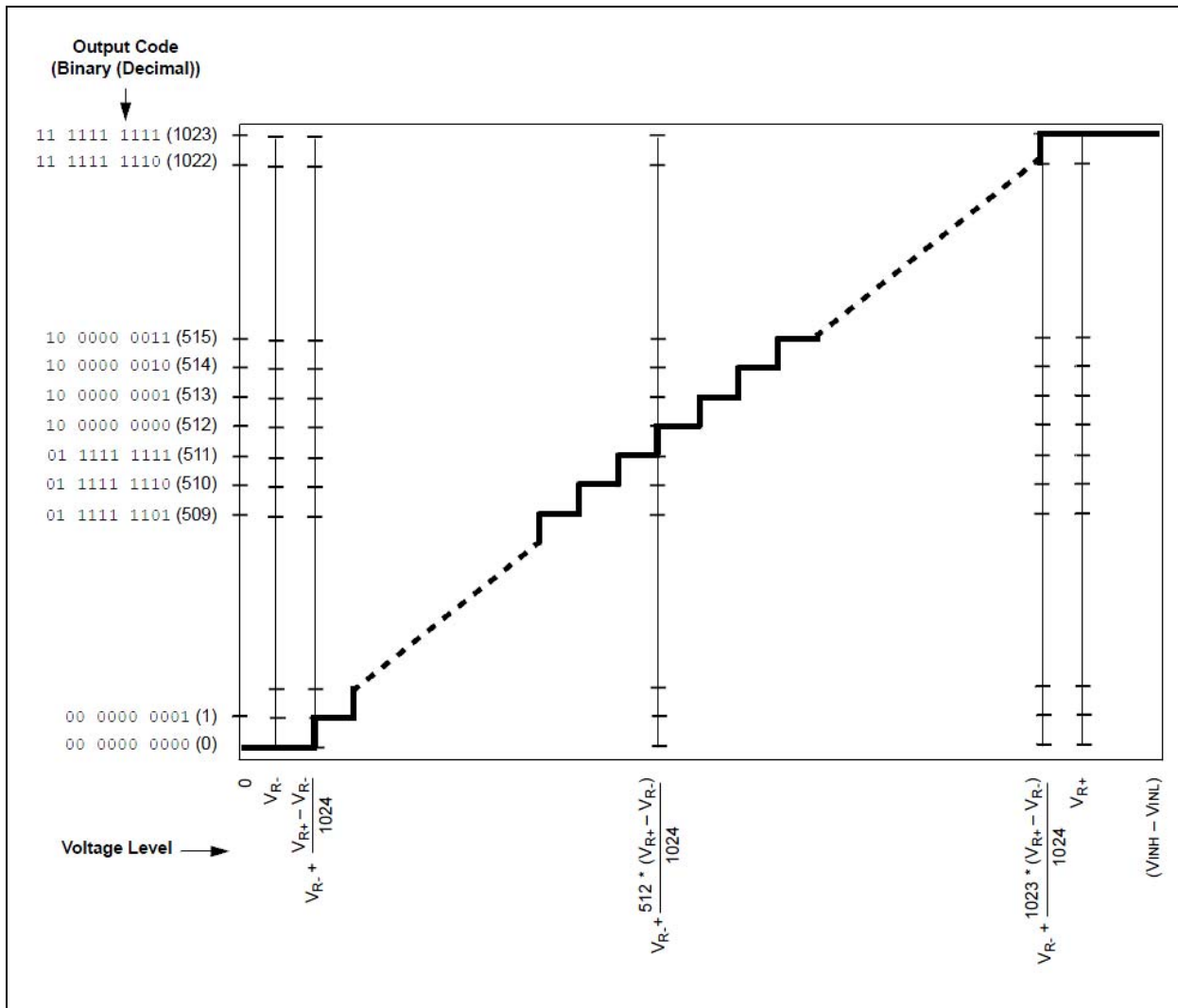
$$TSMP = SAMC<4:0> \cdot TAD$$

$$SAM<4:0> = TSMP/TAD$$

6.2.5 A/D Converter Transfer Function

The 10-bit A/D Converter has 1024 steps. The output is from 0 to 1023. Where 0 is 0.000V, and 1023 is 3.300V for VR+ = AVSS (3.3V) and VR- = AVDD (GND).

FIGURE 6-3: ADC TRANSFER FUNCTION



The first code transition occurs when the input voltage is $((VR+) - (VR-))/1024$ or 1 LSB

- Output 1 is centered at $VR- + (1.5 \cdot ((VR+) - (VR-))/1024)$
- Output 512 is centered at $VREFL + (512.5 \cdot ((VR+) - (VR-))/1024)$
- An input voltage less than $VR- + ((VR-) - (VR-))/1024$ converts as 0
- An input voltage greater than $(VR-) + (1023((VR+) - (VR-))/1024)$ converts as 1023

6.3 THE LAB

6.3.1 Objective

This lab is similar to the previous Comparator Lab. However, this time we will use the 10-bit A/D Converter module to create a more accurate voltage meter.

The 10-bit A/D Converter has an array of timing and control selections, which allows the user to create flexible scanning sequences. Conversions can be started individually by program control, continuously free running, or triggered by selected hardware events.

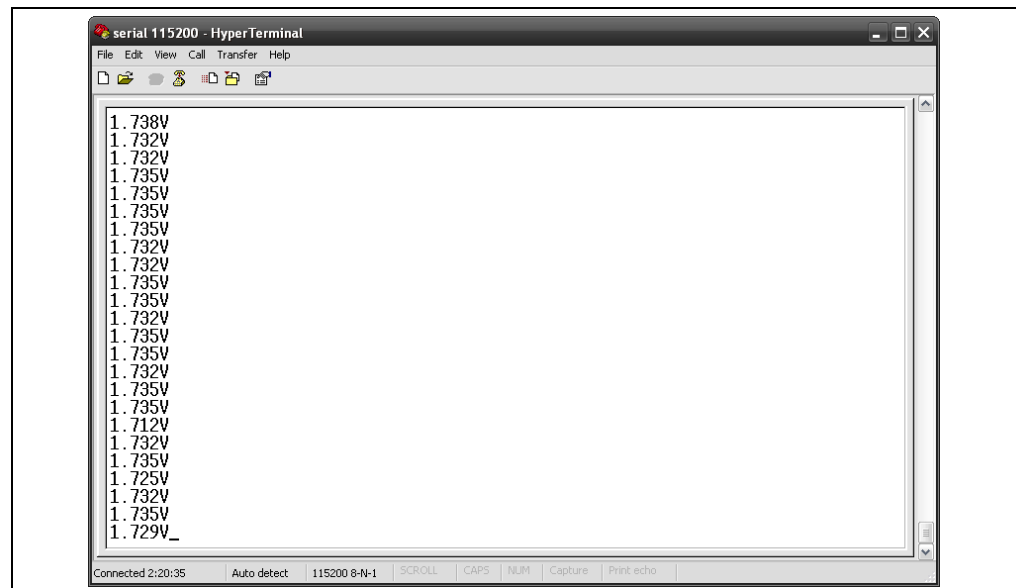
This lab is flexible in allowing you to configure the A/D Converter module any way you choose to complete the task.

This lab including program flow chart assumes the A/D Converter “auto-conversion trigger” and the “ADC1 Conversion Done” interrupt is being used.

6.3.2 Setup

- ADC input to AN4 (pin 21), which is the same input that was used for the Comparator Lab.
- $VR+ = AVSS$
- $VR- = AVDD$
- Setup the 10-Bit A/D Converter module to measure an input voltage on AN4.
- Then use the UART to send the measured voltage to a terminal program.
- You will be required to do some formatting to the A/D Converter output value and send back the voltage as a four digit number formatted similar to “1.234V”.
- The expected output is illustrated in the [Figure 6-4](#).

FIGURE 6-4: EXAMPLE TERMINAL OUTPUT



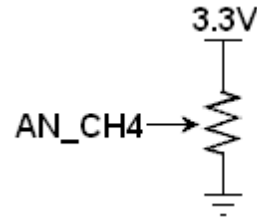
6.3.3 Pertinent Information

You will need to setup a simple variable voltage circuit for the analog input to the A/D Converter. This is the same circuit that was used in Comparator Lab.

Wire a potentiometer as illustrated below.



Potentiometer



Potentiometer Circuit

The potentiometer (also known as a “pot”) is a three terminal resistor that will form an adjustable voltage divider. This will allow adjustment of the voltage at the A/D Converter input between 0 and 3.3V.

6.3.4 Converting the A/D Converter Value

The A/D Converter output value will need to be converted to the corresponding input voltage value. To do this accurately, you will need to use 32bit divisions and multiplications.

A/D Converter step size for VR+ = 3.3V and VR- = GND

$$\text{Step size} = 3.3\text{V}/1023$$

$$\text{Step size} \sim 0.0032258064\text{V}$$

16-bit calculations 216 = 65536

ADC count = 1023 (3.300V)

To convert the value to volts, we can multiply ADC counts by an integer value of step size, but with a 16-bit result register, we are limited to using a value of 32 since a larger integer value would cause the register to overflow.

$$\text{Volts} = 1023 \times 32 = 32736$$

The value 32736 represents 3.2736V.

So, for an ADC output of 1023, which should be 3.3V, we calculate 3.2736V.

32-bit calculations 232 = 4294967296

ADC count = 1023 (3.300V)

To convert the value to volts, we again multiply ADC counts by an integer value of step size, but this time we can use a complete 16-bit value or 32258. This will not overflow the 32-bit results register.

$$\text{Volts} = 1023 \times 32258 = 32999934$$

The value 32999934 represents 3.2999934V.

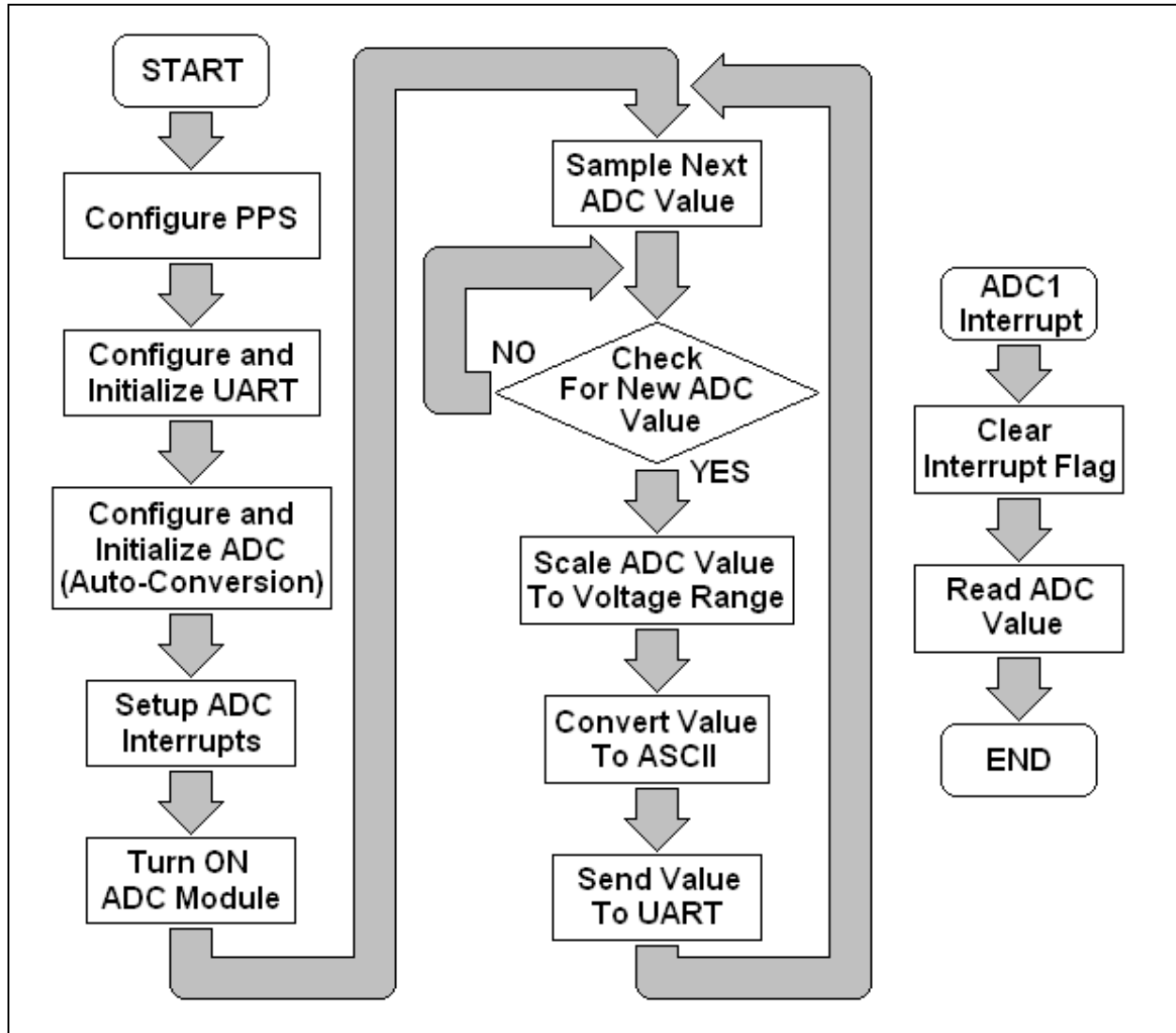
Other methods to convert the A/D Converter value to voltage are possible but this method is advised. This will force the use of 32-bit arithmetic instructions and their associated registers.

Reminder from Lab 2 – Interrupts

__ADC1Interrupt	ADC1 Conversion Done	0x0002E
-----------------	----------------------	---------

The basic program flow is depicted in [Figure 6-5](#).

FIGURE 6-5: LAB 6 PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab6-adc.s`.

LABS

NOTES:

Lab 7. Pulse-Width Modulator (PWM)

7.1 OBJECTIVE

Configure one of the PIC24FJ256GB110 Output Compare Blocks as a PWM to drive an LED and generate a “heart beat” similar to the power LED on a Macintosh computer.

7.2 PRE-LAB

7.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 8. Interrupts (DS39707)
- PIC24F Family Reference Manual – Section 12. I/O Ports with Peripheral Pin Select (PPS) (DS39711)
- PIC24F Family Reference Manual – Section 35. Output Compare with Dedicated Timer (DS39723)

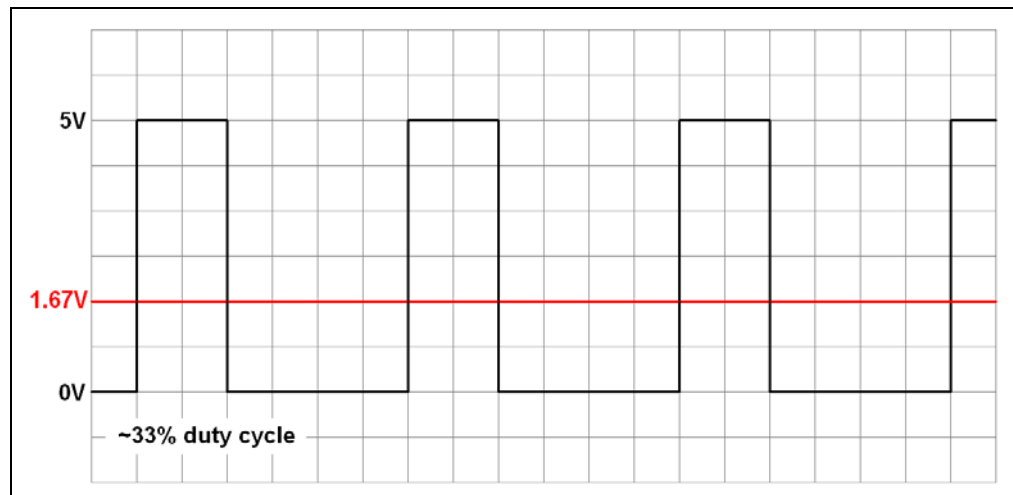
7.2.2 Pulse-Width Modulation (PWM) Mode

Pulse-Width Modulation is used to efficiently regulate the amount of electrical power delivered to a load by quickly switching power between fully ON and fully OFF.

The “duty cycle” is the proportion of ON time in the “period” and is often expressed in percentage. 75% duty cycle would be delivering power $\frac{3}{4}$ of the time, and 100% would be fully on.

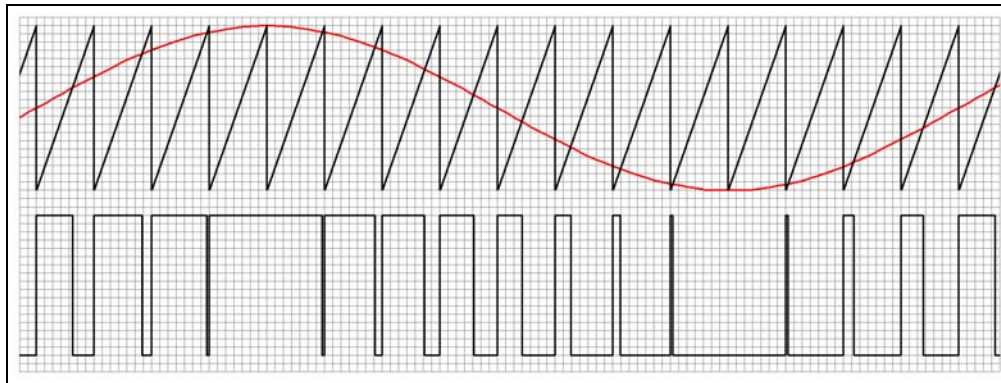
A very simple implementation can be used to deliver an average voltage of 1.67V by pulse-width modulating 5V with a 33% duty cycle as illustrated below.

FIGURE 7-1: BASIC PWM TO CONTROL POWER DELIVERY



A more complicated implementation can be used to deliver sinusoidal power. This is a similar example to what we will be doing in this lab.

FIGURE 7-2: USING PWM TO CREATE A SINE WAVE



7.2.3 Output Compare Blocks

The PIC24FJ256GB110 has 9 Output Compare Blocks. Each of these blocks can be configured as any of the following:

- Center-Aligned PWM mode
- Edge-Aligned PWM Mode
- Double Compare Continuous Pulse mode
- Double Compare Single-Shot mode
- Single Compare mode
- Single Compare Single-Shot mode

In PWM mode, the output compare module can be configured for edge-aligned or center-aligned pulse waveform generation. All PWM operations are double-buffered (buffer registers are internal to the module and are not mapped into SFR space).

To configure the output compare module for PWM operation:

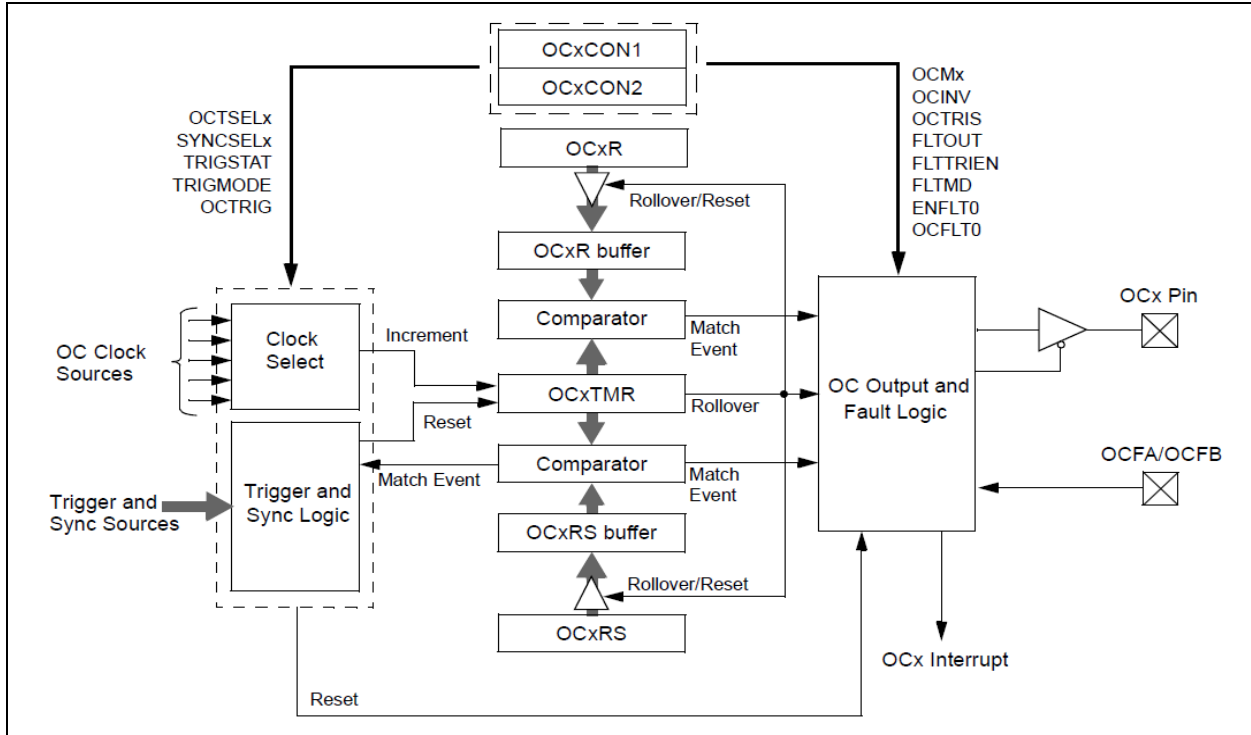
1. Configure the OCx output for one of the available Peripheral Pin Select pins.
2. Calculate the desired duty cycles and load them into the OCxR register.
3. Calculate the desired period and load it into the OCxRS register.
4. Select the current OCx as the sync source by writing 0x1F to SYNCSEL<4:0> (OCxCON2<4:0>), and clearing OCTRIG (OCxCON2<7>).
5. Select a clock source by writing the OCTSEL<2:0> (OCxCON<12:10>) bits.
6. Enable interrupts, if required, for the timer and output compare modules. The output compare interrupt is required for PWM Fault pin utilization.
7. Select the desired PWM mode in the OCM<2:0> (OCxCON1<2:0>) bits.
8. If a timer is selected as a clock source, set the TMRy prescale value and enable the time base by setting the TON (TxCON<15>) bit.

This peripheral contains input and output functions that may need to be configured by the Peripheral Pin Select.

Registers associated with the Output Compare Block when used as a PWM module:

- OCxCON1: Output Compare x Control Register 1
- OCxCON2: Output Compare x Control Register 2
- OCxR: Compare Register
- OCxRS: Secondary Compare Register

FIGURE 7-3: OUTPUT COMPARE BLOCK DIAGRAM



The OCx outputs must be assigned to an available RPN pin before use.

7.2.4 PWM Duty Cycle

The PWM duty cycle is specified by writing to the OCxRS and OCxR registers. The OCxRS and OCxR registers can be written to at any time, but the duty cycle value is not latched until the period is complete. This provides a double buffer for the PWM duty cycle and is essential for glitchless PWM operation.

Some important boundary parameters of the PWM duty cycle include:

- If OCxR, OCxRS, and PRy are all loaded with 0000h, the OCx pin will remain low (0% duty cycle).
- If OCxRS is greater than PRy, the pin will remain high (100% duty cycle).

7.2.5 Edge-Aligned PWM Mode Operation

When synchronization occurs, the following five events occur on the next increment cycle:

- The timer is reset to zero and resumes counting
- The OCx pin is set high (exception: if OCxRS = 0000, the OCx pin would not be set)
- The OCxR and OCxRS Buffered registers are updated from OCxR and OCxRS
- Interrupt flag, OCxIF, is set
- When the timer and OCxR match, the pin would be set low. This match does not generate interrupts.

Figure 7-4 depicts the output compare pin over time with OCxR = 0x0000 and OCxRS = 0x5000.

FIGURE 7-4: OUTPUT COMPARE WITH INITIAL OCxRS = 0x0000

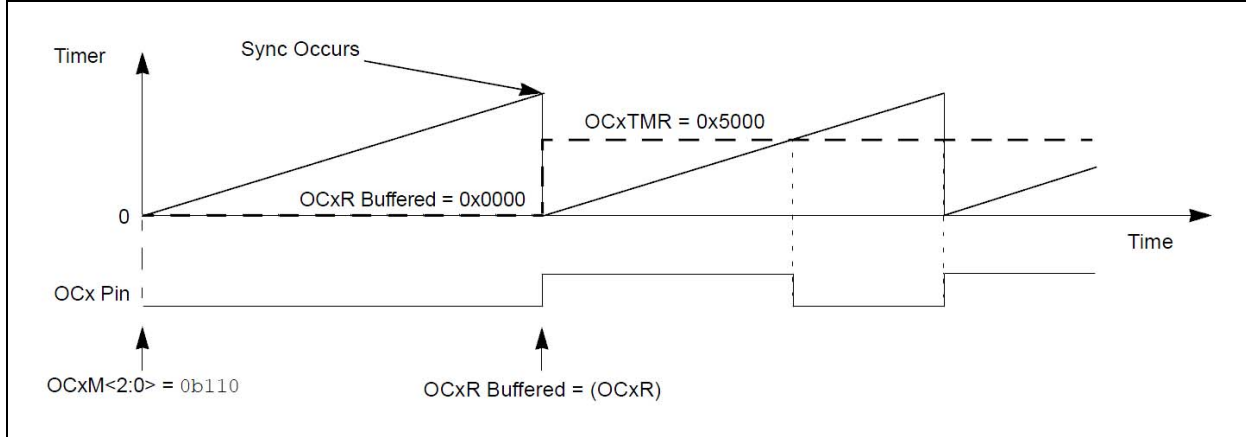
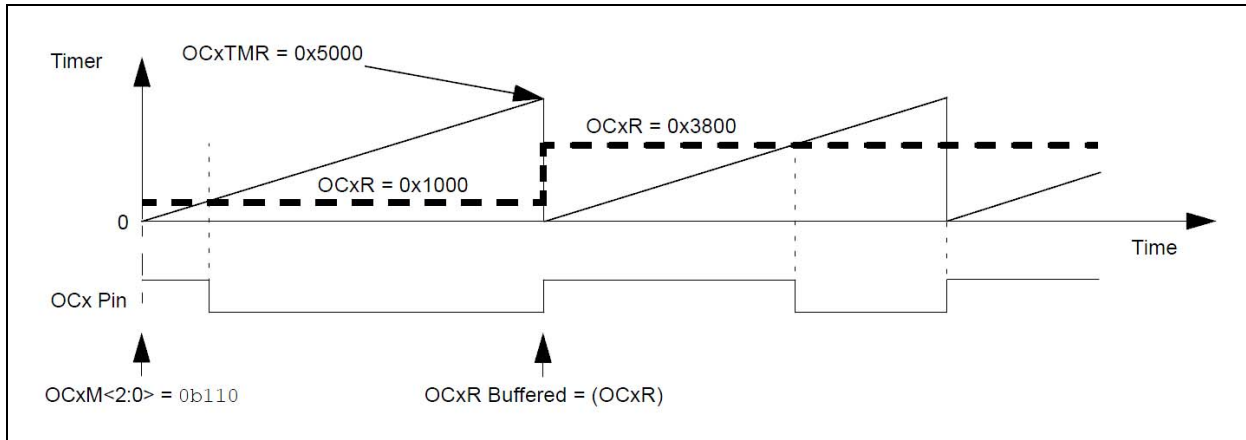


Figure 7-5 depicts the output compare pin over time with OCxR = 0x1000 and OCxRS = 0x5000.

FIGURE 7-5: OUTPUT COMPARE WITH INITIAL OCxRS = 0x1000



7.3 THE LAB

7.3.1 Objective

Write code to configure and enable one of the nine PIC24FJ256GB110 output compare blocks as a PWM. Use the PWM to generate an LED “heart beat” similar to the power LED on a Macintosh computer.

You will need to generate a similar pattern to the sinusoid illustrated earlier. Although generating a sinusoid is possible, it is somewhat unnecessary. A triangle like wave with some tweaking should work sufficiently for our intended purpose. The LED should “beat”. It should repeatedly slowly transition from OFF to ON and then back to OFF.

7.3.2 Pertinent Information

The LEDs are NOT connected to remappable peripheral output pins. Therefore, it will not be possible to drive the LED directly with the PWM output. Rather it will be necessary to toggle the desired LED port pin based on the PWM output.

LEDs are on the following ports:

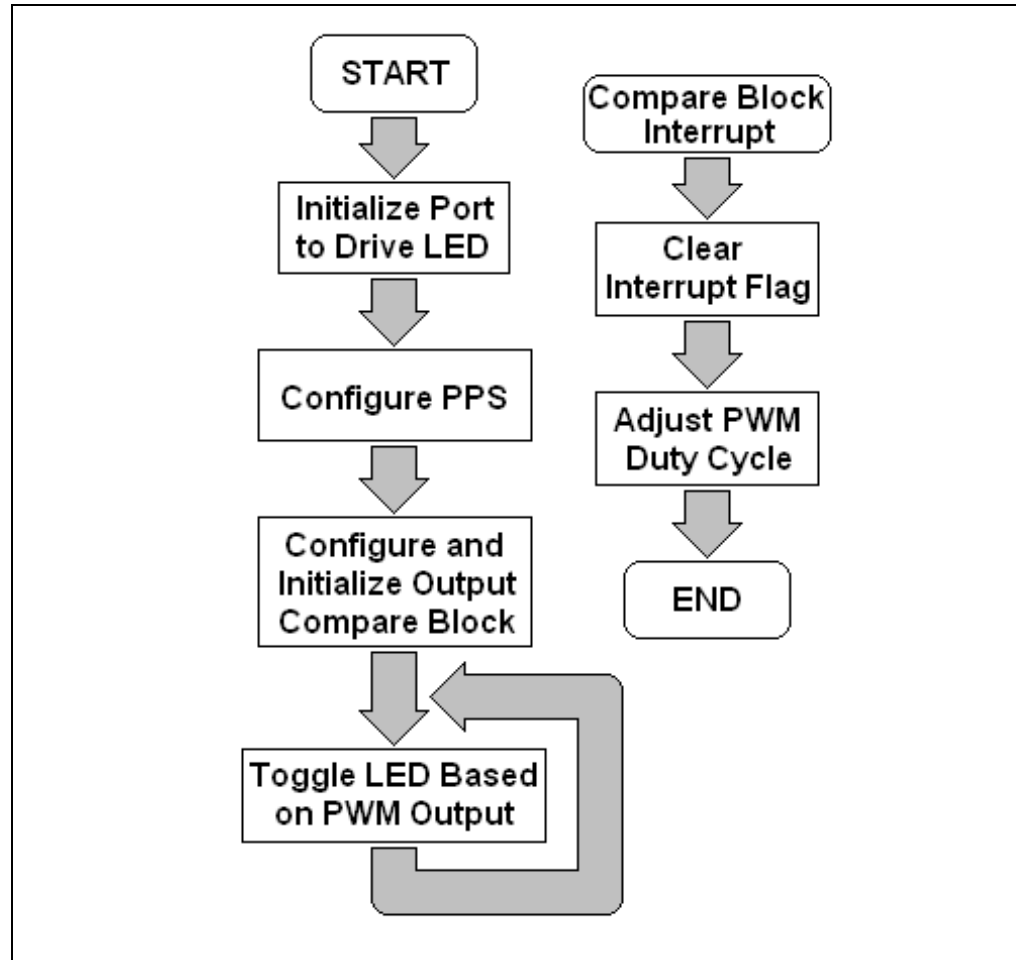
LED	PIC® MCU Port
LED1	Port C Bit 2
LED2	Port C Bit 3
LED3	Port C Bit 4
LED4	Port E Bit 9

Reminder From lab2-interrupts:

__OC1Interrupt	Output Compare 1	0x00018
__OC2Interrupt	Output Compare 2	0x00020

The basic program flow is depicted in [Figure 7-6](#).

FIGURE 7-6: LAB 7 BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab7-pwm.s`.

LABS

NOTES:

Lab 8. External Memory

8.1 OBJECTIVE

Investigate the MX PIC24F module memory resources and the general design issues surrounding memory use in embedded systems.

8.2 PRE-LAB

8.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- MPLAB IDE User's Guide (DS51519)
- PIC24F Family Reference Manual – Section 23. Serial Peripheral Interface (SPI) (DS39699)

8.2.2 Memory Basics

At a basic level, memory circuits hold their contents for future accesses. Most memory provides an interface to write information into the memory (i.e. store) and to read information out of the memory (i.e. load). The write interface is not always available during run time, e.g. One-Time-Program (OTP) memories.

The ability of a memory circuit to hold its contents when power is removed is one way to categorize memory. The contents of volatile memory are erased when power to the circuit is disconnected. Examples of this are the DDR memory of a desktop computer. Nonvolatile memory holds its contents even when power is removed from the circuit. Examples of this are a USB disk that contains NAND Flash memory.

8.2.3 Memory in Embedded Systems

In some ways, developing firmware for an embedded controller is not unlike writing code for a desktop computer. Many aspects of the design process are similar: languages can often be the same (i.e. "C"), the control loops are similar, the sequential access is the same, and even multi-tasking and threads can be used on each. But when it comes to memory, the equations are completely different.

A desktop computer has resources that are simply not feasible for embedded systems, e.g. a large hard drive. As a result, most modern OSs assume that a large hard drive is connected to the computer. By using a technique called "virtual memory", a program that is executing has access to an almost unlimited amount of memory. If the OS needs more memory, then it will swap out little used areas of its volatile memory to the hard drive. This then allows the OS to use that volatile memory space for further program execution. If the memory that was previously swapped out is required, then the OS will swap it back with another area of memory. As a result, most programmers can assume that they have a virtually unlimited amount of RAM memory to use, and the OS takes care of the details.

Most embedded systems do not have such features. This is primarily due to the size, cost, and power consumption required to have such complicated hardware. As a result, designers of embedded systems must constantly evaluate the consumption of the limited resources of the microcontroller. If a program runs out of memory, there is no recovery and the system will fail. In some cases it will not even compile.

8.2.4 PIC24F Memory

The PIC24FJ256GB110 family has the following internal memory resources:

Memory	Size	Classification
Program Memory	256 Kbytes	Nonvolatile
RAM	16 Kbytes	Volatile
Date EEPROM	0	Nonvolatile

Note 1: Some of the newer PIC24F_K devices do incorporate internal EEPROM.

It is possible to use the program memory to store data in a nonvolatile fashion. If the data being stored is only going to be written a small number of times, this is sufficient. However, since the program memory is only guaranteed up to about 1000 write cycles, this is not sufficient for data that will be written many times over a long lifespan.

EEPROM is typically guaranteed for many more write cycles. 10,000; 100,000; or even 1,000,000 are typical specifications. Since the PIC24F on board does not have any internal EEPROM memory, this is provided by an external memory chip.

8.2.5 Extending Microcontroller Resources

No matter how many internal resources a microcontroller has, there is a real-world application that needs more or different resources. Examples of this would be EEPROM memory or some sort of specific sensor circuit like a temperature sensor.

In these cases, it is necessary to connect an external chip or circuit. To facilitate this, microcontroller manufacturers provide standard bus interfaces. The PIC24FJ256GB110 is no exception, providing three SPI bus controllers and three I²C bus controllers, as well as the Parallel Master Port (PMP) module. There are many external circuits and sensors that can be connected on these busses to extend and enhance the functionality of the microcontroller.

8.2.6 MX PIC24F EDU Module Memory

The MX PIC24F EDU Module provides the following external memory resources that are available to the PIC24F.

Memory	Size	Classification
External EEPROM	32 Kbytes	Nonvolatile

This external EEPROM is guaranteed to retain its data for more than 200 years over 1,000,000 write cycles!

Access to the external EEPROM memory is provided using the SPI bus. The chip select signal, SPI_CS_ROM, is connected to pin RB13 of the PIC MCU.

8.2.7 Serial Peripheral Interface (SPI)

SPI was created by Motorola. The SPI bus, also referred to as the “four wire” bus, is a single master multiple slave full duplex synchronous serial interface. The SPI operates with a single master and one or more slave devices. Only one device can talk to the master at any given time. Therefore, a chip select or “Slave Select (SS)” is required from the master to each slave device. When more than one slave device is connected to the SPI bus, these slave devices are required to tri-state their MISO output when they are not selected. Devices without tri-state outputs cannot share an SPI bus and therefore can only be used in a single slave configuration.

The master pulls the Slave Select Low for the desired Slave device and sends a bit on the MOSI line. The master must then wait if the slave device requires a wait period, which is often the case for Analog-to-Digital Converters.

Next, the master issues clock cycles. A full duplex transmission occurs during each clock cycle. The master drives the clock line high. On the rising edge of the clock, the slave reads in the bit on the MOSI line and shifts out the MSB from its shift register onto the MISO line. The master drives the clock low and reads the MISO line and places its next bit on the MOSI line. The master drives the clock high and the clock cycle repeats.

Note: The master must clock in an extra eight dummy bits to read the last eight bits from the slave.

FIGURE 8-1: SIMPLIFIED SPI SYSTEM

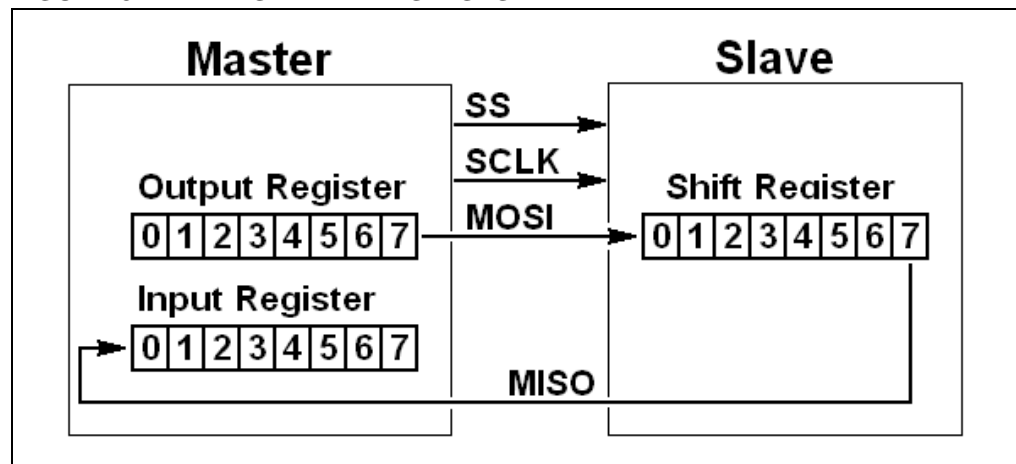
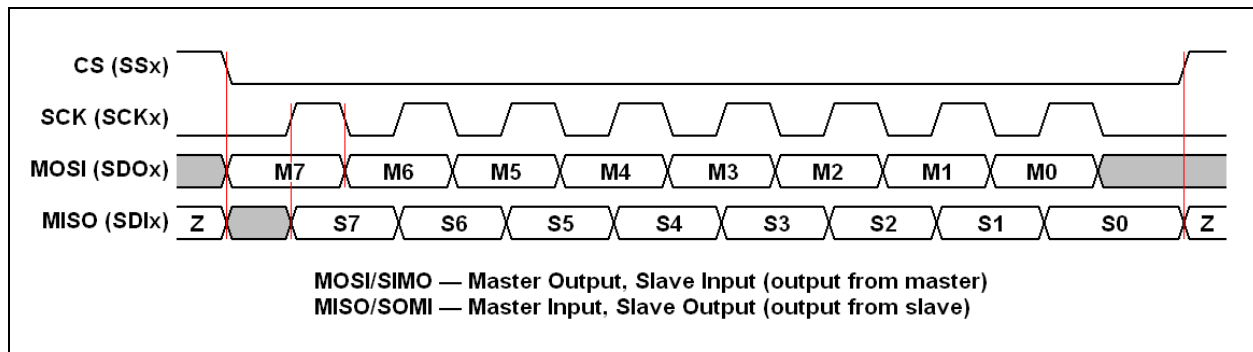


FIGURE 8-2: GENERIC SPI TIMING



The PIC24FJ256GB110 has three SPI modules. The SPI serial interface consists of four pins:

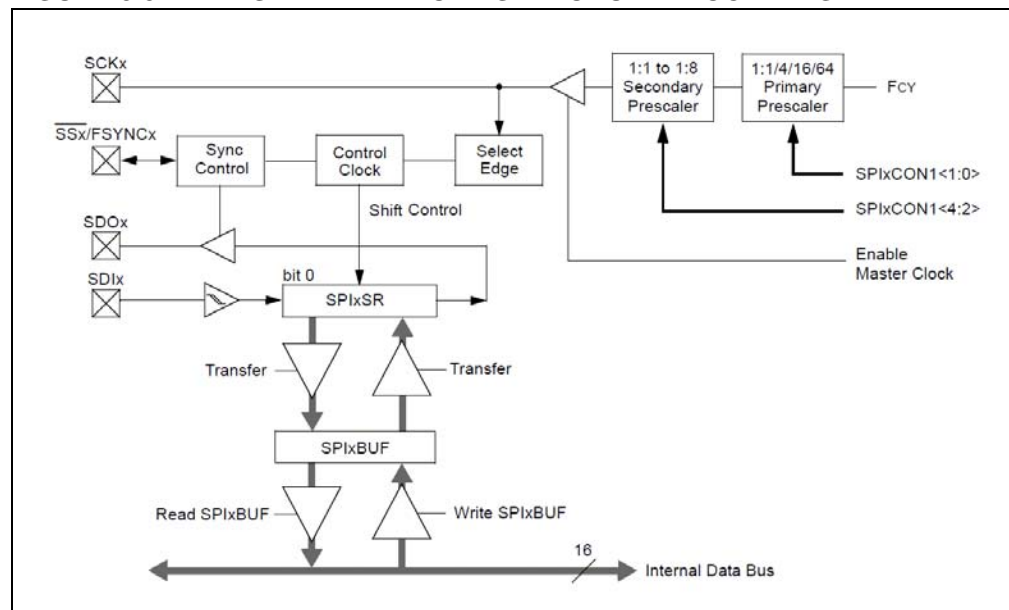
- SDIx: Serial Data Input
- SDOx: Serial Data Output
- SCKx: Shift Clock Input or Output
- SSx: Active-Low Slave Select or Frame Synchronization I/O Pulse

The SPI module can be configured to operate using 2, 3 or 4 pins.

In 3-pin mode, SSx is not used. In 2-pin mode, both SDOx and SSx are not used.

The PIC24FJ256GB100 SPI module provides two methods of operation: Standard and Enhanced Buffer mode. In Standard mode, data is shifted through a single serial buffer. In Enhanced Buffer mode, data is shifted through an 8-level FIFO buffer. In this lab the Standard mode is sufficient. The Enhanced Buffer mode is useful when higher data rates are required.

FIGURE 8-3: STANDARD MODE SPI MODULE BLOCK DIAGRAM



To set up the SPI module for the Standard Master mode of operation:

1. If using interrupts:
 - a) Clear the SPIxIF bit in the respective IFS register.
 - b) Set the SPIxIE bit in the respective IEC register.
 - c) Write the SPIxIP bits in the respective IPC register to set the interrupt priority.
2. Write the desired settings to the SPIxCON1 and SPIxCON2 registers with MSTEN (SPIxCON1<5>) = 1.
3. Clear the SPIROV bit (SPIxSTAT<6>).
4. Enable SPI operation by setting the SPIEN bit (SPIxSTAT<15>).
5. Write the data to be transmitted to the SPIxBUF register. Transmission (and reception) will start as soon as data is written to the SPIxBUF register.

8.3 THE LAB

8.3.1 Objective

Write a program to look for a string at a particular address in the external EEPROM. Then, using the UART and a terminal emulator like in Lab 3, display the string or an error message. Finally, provide a simple prompt that allows the user to enter a new string. This string, if entered, would be stored into the same location in external memory.

Record how much program and data memory your program takes. This is reported by MPLAB IDE in the menu *View>Memory Usage*.

8.3.2 Pertinent Information

In order to write into EEPROM, it must first be erased. Then, and only then, can it be successfully programmed. An EEPROM that has been erased will read 0xff as the value of each byte. This is how the program will know if a string has been entered since 0xff is not a valid ASCII character.

The string could either be of fixed length or could be NULL terminated (like C uses). You will have to add the logic for the NULL termination to both the writing and reading of the data, i.e. you will have to write an extra 0x00 value after the string entered by the user.

When enabling the SPI for the external 25LC256 EEPROM memory, you will have to drive the SPI_CS_ROM (pin 42) and SPI_CS_ID (pin 92) directly and run the SPI in 3-pin mode. It is important to ensure that other devices on the SPI bus are inactive.

SPI chip select lines:

SPI_CS_ROM (pin 42), PORTB (pin 13)

You will have to configure the PPS for the SPI for the following pins:

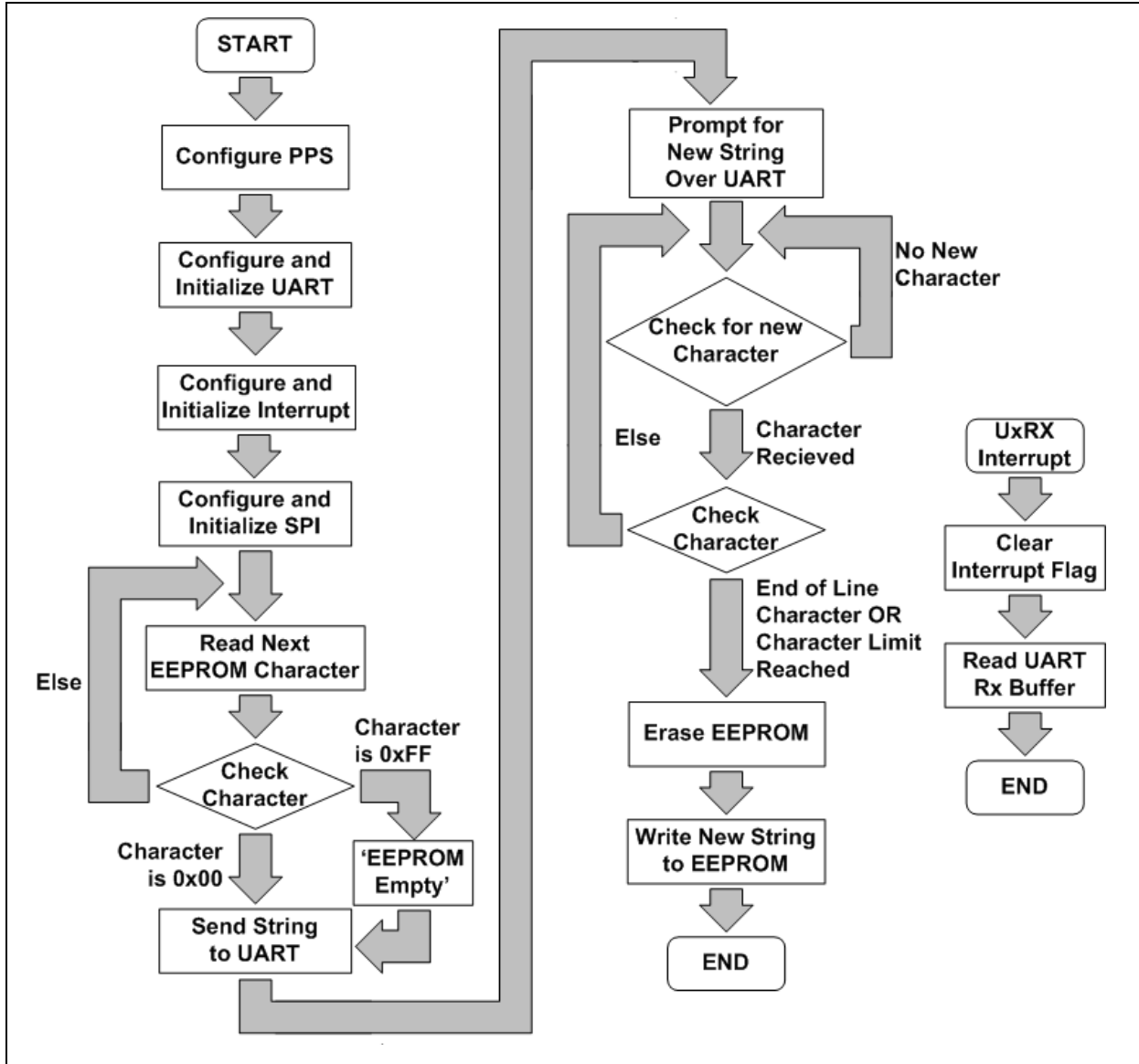
SPI_SCK (pin 10), configure PPS output RP21 to SCKxOUT

SPI_MOSI (pin 12), configure PPS output RP19 to SDOx

SPI_MISO (pin 11), configure PPS input RP26 to SPIx

The basic program flow is depicted in [Figure 8-4](#).

FIGURE 8-4: BASIC PROGRAM FLOW



The template assembly file sets up this basic program flow. The student will need to add the appropriate code to the relevant sections of the file `lab8-memory.s`.

Lab 9. Power Management

9.1 OBJECTIVE

Learn to consider power consumption issues for embedded designs.

9.2 PRE-LAB

9.2.1 Reference Material

- PIC24FJ256GB110 Family Data Sheet (DS39897)
- 16-Bit MCU and DSC Programmer's Reference Manual (DS70157)
- PIC24F Family Reference Manual – Section 6. Oscillator (DS39700)
- PIC24F Family Reference Manual – Section 10. Power Saving Features (DS39698)
- PIC24F Family Reference Manual – Section 14. Timers (DS39704)
- AN1267 nanoWatt and nanoWatt XLP Technologies: An Introduction to Microchip's Low-Power Devices (DS01267)

9.2.2 Power

Every electronic system requires power to run. For systems powered by batteries, it is important to minimize the amount of power required – this extends battery life or allows for the use of a smaller battery. As the number of electronic systems continues to grow, it is becoming more and more important to reduce the power consumption and the impact on our planet. The concepts presented in this lab are a first step to understanding this complex topic.

There are two components to the total power consumption of a system: static power consumption and dynamic power consumption.

Static power is the current consumed when the main clock is disabled. In general, this includes supervisory circuitry as well as logic required to turn the main clock on due to some external condition. This also includes the leakage currents of the device transistors themselves. In general, this is very low and fixed for a given microcontroller device.

Dynamic power is the current consumed by the switching of digital logic. The higher the frequency, the more switches occur in a given amount of time and more power is consumed. Dynamic power tends to be much larger than static power for embedded systems and the design of the software can greatly influence this factor. Thus, this lab will focus on methods to reduce the dynamic power.

9.2.3 Calculating Power

The dynamic power of a circuit can be calculated with the following equation:

$$P_{DYN} = C \cdot V^2 \cdot F$$

Where PDYN is the power consumed in Watts, C is the capacitance being switched per clock cycle in Farads, V is the amplitude of the voltage in Volts, and F is the frequency of the switching in Hz. This equation is for one switched node. The total dynamic power consumed by a circuit is simply the sum of the dynamic power consumed for each node in the circuit.

The capacitance of a node is determined by the fabrication of the chip and the printed circuit board circuit. From a software perspective this power consumption is fixed.

The voltage amplitude of the switched signal has the largest impact on the dynamic power consumed as it obeys the square law. Reducing the voltage of the system greatly reduces the dynamic power. This again is a system level design choice and is typically not within the domain of software control.

The clock switching frequency is the one factor that is heavily controlled by the software running on a microcontroller. Microcontrollers like the PIC24F that have been designed with low power in mind provide multiple facilities to reduce the dynamic power consumption of the microcontroller.

It is critical to note that to obtain the lowest power consumption of an embedded system, all of these items need to be considered at the beginning of a design cycle. The Microchip PIC24F and the MX module have already determined the capacitance of each node and the voltage amplitude of each interface. This lab will focus on reducing the frequency of switching.

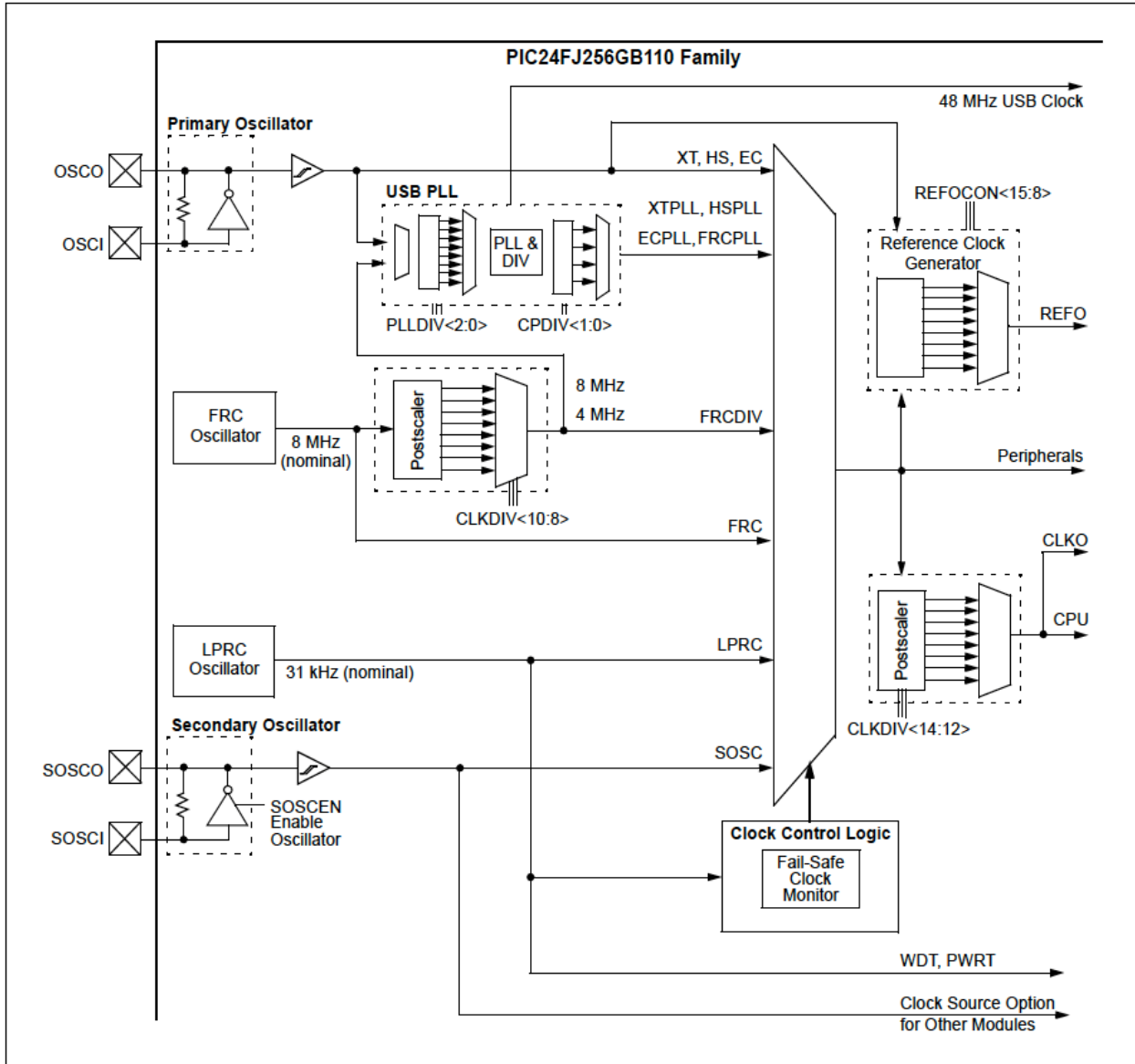
9.2.4 Reducing Power – Main Clock

The discussion in [Section 9.2.3 “Calculating Power”](#) highlighted that the main method to reduce power consumption is to reduce the number of times the main clock is switching. The primary method of doing this is to control the main clock frequency. Microchip’s PIC24F provides many methods to do this.

Section 8.0 of the PIC24FJ256BG110 Family Data Sheet describes all the details of the oscillator configuration. [Figure 9-1](#) is reproduced as a quick reference. The main system clock (labeled CPU) is controlled by many configuration registers and control registers and can take on many different frequencies.

In general, a system’s main clock should be run at the slowest speed that allows for acceptable system performance. However, for systems that have bursts of activity followed by long idle times, it may be advantageous to run at a higher clock frequency. A higher clock frequency allows the calculations to be completed faster, and the system can get into its lowest power consumption mode faster. The trade-off for this situation is unique to each system.

FIGURE 9-1: PIC24FJ256GB110 FAMILY CLOCK DIAGRAM



9.2.5 Reducing Power – Turn Off Unused Peripherals

While turning off the main clock or greatly reducing the main clock frequency is the most effective power consumption strategy, it often isn't the only solution required. If the main clock is turned off, then the system can't actually do anything other than wake up, which doesn't make for a very useful system. In order to reduce power consumption, it is necessary to turn off the clock to sub-circuits that are not being used. If $F = 0$ Hz (i.e. not switching), then no dynamic power will be consumed for a given sub-circuit.

Microchip provides a few operating modes in order to help with this. Each operating mode provides the clock to a different subset of internal peripheral circuits. [Table 9-1](#) from AN1267 lists the different Operating Modes and what peripherals are active in each mode. It also describes the current consumption in each mode and how the system can exit the mode (or wake-up).

Section 9.0 of the PIC24FJ256GB110 Family Data Sheet provides the details of the specific power-saving features of this device.

TABLE 9-1: MICROCHIP PIC[®] MCU OPERATING MODE OVERVIEW

Operating Mode	Active Clocks	Active Peripherals	Wake-up Sources	Typical Current	Typical Usage
Deep Sleep ⁽¹⁾	<ul style="list-style-type: none"> • Timer1/SOSC • INTRC/LPRC 	<ul style="list-style-type: none"> • RTCC • DSWDT • DSBOR • INT0 	<ul style="list-style-type: none"> • RTCC • DSWDT • DSBOR • INT0 • $\overline{\text{MCLR}}$ 	< 50 nA	<ul style="list-style-type: none"> • Long life, battery-based applications • Applications with increased Sleep times
Sleep	<ul style="list-style-type: none"> • Timer1/SOSC • INTRC/LPRC • A/D RC 	<ul style="list-style-type: none"> • RTCC • WDT • ADC • Comparators • CVREF • INTx • Timer1 • HLVD • BOR 	All device wake-up sources (see device data sheet)	50-100 nA	Most low-power applications
Idle	<ul style="list-style-type: none"> • Timer1/SOSC • INTRC/LPRC • A/D RC 	All Peripherals	All device wake-up sources (see device data sheet)	25% of Run Current	Any time the device is waiting for an event to occur (e.g., external or peripheral interrupts)
Doze ⁽²⁾	All Clocks	All Peripherals	Software or interrupt wake-up	35-75% of Run Current	Applications with high-speed peripherals, but requiring low CPU use
Run	All Clocks	All Peripherals	N/A	See device data sheet	Normal operation

Note 1: Available on PIC18 and PIC24 devices with nanoWatt XLP Technology only.

Note 2: Available on PIC24, dsPIC[®] DSC and PIC32 devices only.

9.2.6 Reducing Power – Go to Sleep

For many embedded systems, most of their operating life is spent waiting for a human to interact with the system (press a button for instance). Until this interaction is detected there is very little for the system to do. These are prime opportunities to go into the Sleep operating mode (i.e., turn off the main clock).

[Table 9-1](#) lists INT0 as a possible wake-up source. This means that the system can go into the Deep Sleep Mode (lowest power consumption possible) and still monitor the INT0 input. If the system can be designed such that the human interaction triggers this interrupt, then the system will consume the least amount of dynamic power.

9.2.7 Microchip XLP Technology

As more electronic applications require low power or battery power, energy conservation becomes paramount. Today's applications must consume little power and, in extreme cases, last for up to 15-20 years, while running from a single battery. To enable applications like these, products with Microchip's nanoWatt XLP Technology offer the industry's lowest currents for Run and Sleep, where extreme low-power applications spend 90%-99% of their time.

Benefits of nanoWatt XLP Technology:

- Sleep currents below 20 nA
- Brown-out Reset down to 45 nA
- Watchdog Timer down to 220 nA
- Real-time Clock/Calendar down to 470 nA
- Run currents down to 50 μ A/MHz
- Full analog and self-write capability down to 1.8

Visit www.microchip.com/xlp for more information.

9.3 THE LAB

9.3.1 Objective

Using the Lab 2 framework, use the power reduction concepts of the pre-lab to reduce the total power consumption of the system. Measure the power consumed to confirm that the changes are effective in reducing power.

9.3.2 Pertinent Information

The clock setup is configured in both configuration bits (set during programming and specified in file `config_bits_pic24fj256gb110.inc`) and control register bits (dynamically set by the application code at run time).

If the main clock frequency is changed, then the timer pre-scaler settings may need to change.

Use a voltmeter to measure the voltage across the input power measurement resistor (R32). This is a 0.1 Ohm resistor, and by using the equation $P = VI$, the total current consumed can be calculated. Do this for the project before and after the power consumption changes are made and record your results.

LABS

NOTES:

Lab 10. Project

10.1 OBJECTIVE

Simulate a real embedded system.

10.2 THE LAB

10.2.1 Objective

Use the I/O of the MX PIC24F EDU Module and the MX Educational Target to simulate a real-life system. Write a specification that defines which I/O is used to simulate the environment and then write the code to control the system accordingly.

Use at least 5 of the lab topics in the system.

10.2.2 Elevator Control Example

10.2.2.1 SYSTEM OVERVIEW

The system to be developed is a 4-floor elevator controller. These will be called Level 1-Level 4. Each level will have a request button that users would press to request service from the elevator. Each level will also have an indicator to display that service has been requested. Once the request is serviced, the indicator will be turned off.

Inside the elevator, a character display will inform the occupants of the floor that the elevator cart is approaching.

10.2.2.2 I/O MAPPING

The 4 push buttons on the MX Educational Target will represent the request buttons for each level of the building. The 4 LEDs on the MX Educational Target will represent the current location of the elevator.

The UART will be used to output the text to a terminal emulator running on an external PC. The terminal emulator should be set to 9600 baud 8-N-1 to display the text properly. The UART will also be used to input the destination floor for the user.

The timer interrupt will be used to simulate the time it takes for the elevator cart to go from one floor to the next. This time will be set to 3 seconds from one floor to the next floor.

10.2.2.3 TOPICS COVERED

1. I/O port control from Lab 0
2. UART control from Lab 3
3. Timers from Lab 1
4. Interrupts from Lab 2
5. Power-saving features from Lab 9

10.2.2.4 FUNCTIONAL DESCRIPTION

The system will power up and assume that it is on level 1. The service request indicators on each level will be turned off. The state will default to Idle Request.

The elevator cart controller has four states: Idle Request, Moving Request, Idle Target, and Moving Target. The Request states are a result of the service request buttons, and the Target states are a result of the target being entered from the terminal emulator over the UART.

When a request button is pressed, the service request indicator will be turned on. When in Idle state, a request from any level (except the current level) will cause the state to change to the Moving Target state. The elevator will move in the correct direction given its current location and the location of the service request. The elevator will move to the level of the requested level, turn off the service requested indicator, and then move to the Idle Target state. Subsequent button presses on service request buttons will be ignored until the current service is completed.

When in Idle Target state, the system is waiting for the user to enter the desired target level. When a valid number is entered, the system will then move to the desired level. When this movement is completed, the system will return to Idle Request, at which time subsequent service requests will be processed.

At all times the level display on the terminal emulator will indicate the current level (if in Idle Request or Idle Target state), or the level that is being approached during movement.

10.2.2.5 POSSIBLE ENHANCEMENTS TO FUNCTIONALITY

There are many ways this system could be made better. Consider adding these functional enhancements:

- Queueing of service requests
- Simulating ramping up and down speed of the elevator cart
- Blinking level indicator when that level is the target level and the cart is approaching
- ...

Appendix A. Creating an MPLAB IDE Project

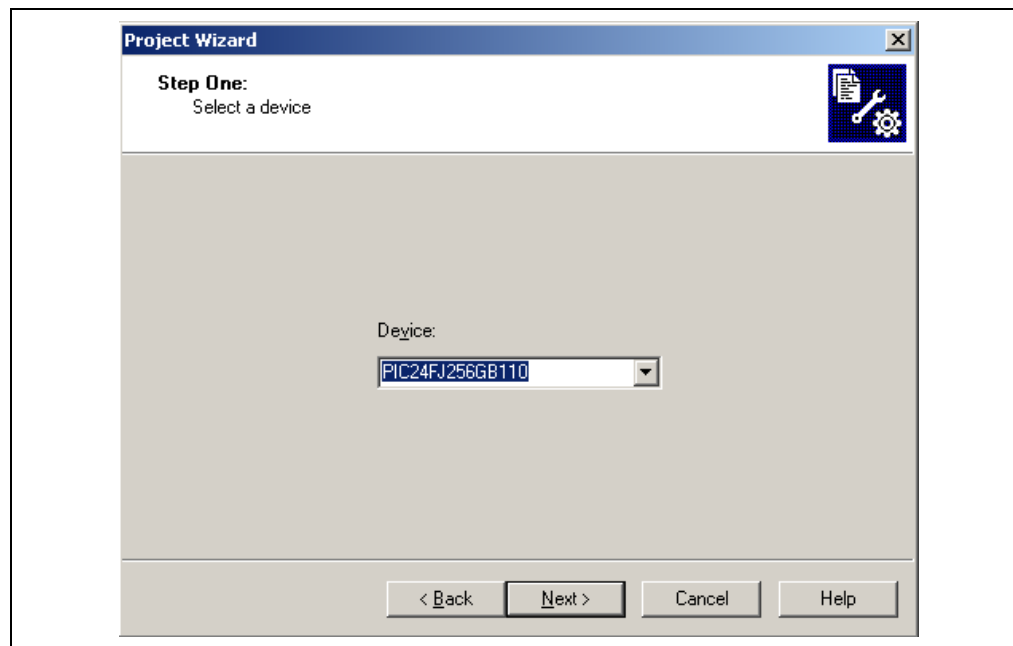
This appendix will walk through the process of creating a new MPLAB IDE project in a step-by-step manner. These labs assume MPLAB IDE version 8.63 is being used. Subsequent versions of MPLAB IDE will most likely follow a very similar process but may differ in small details.

The easiest way to create a new project is to use the Project Wizard. Advanced users can perform this process manually but it is often faster to use the Project Wizard and then simply modify specific characteristics of the project.

A.1 STEP ONE: DEVICE SELECTION

To start, select *Project>Project Wizard...* from the MPLAB IDE menu. This will launch the Project Wizard. Clicking on “Next” on the first screen will begin the process. The Step One dialog is where you select the target device. In the case of these labs, the target device is the PIC24FJ256GB110. Scroll through the drop-down box to select this device as shown in [Figure A-1](#). Click on “Next” to continue.

FIGURE A-1: PROJECT WIZARD STEP ONE



A.2 STEP TWO: LANGUAGE TOOLSUITE

In step two, the software toolset is selected. This consists of the assembler, linker, and optional C language compiler (not used in these labs). In the Active Toolsuite drop-down box, select “Microchip ASM30 Toolsuite”. This toolsuite consists of a collection of three individual tools:

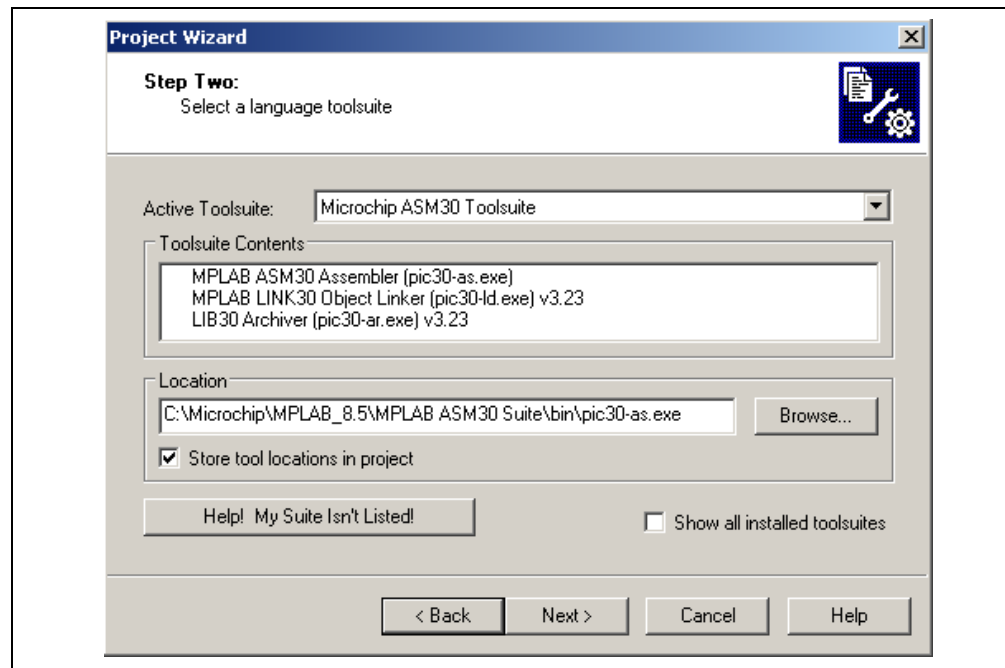
- ASM30 Assembler
- LINK30 Linker
- LIB30 Archiver

In general, the default locations will be correct. In the case where multiple versions of MPLAB IDE are installed on the same computer, it is necessary to ensure that the exact location of each of the three tools points to the correct version (normally the latest). The most important thing is that all three tools point to the same version.

Enable the checkbox “Store tool locations in project” to store these settings in the project files.

Figure A-2 shows a typical dialog for this step. Note that the exact location of the installed tools will vary based on local IT policies.

FIGURE A-2: PROJECT WIZARD STEP TWO

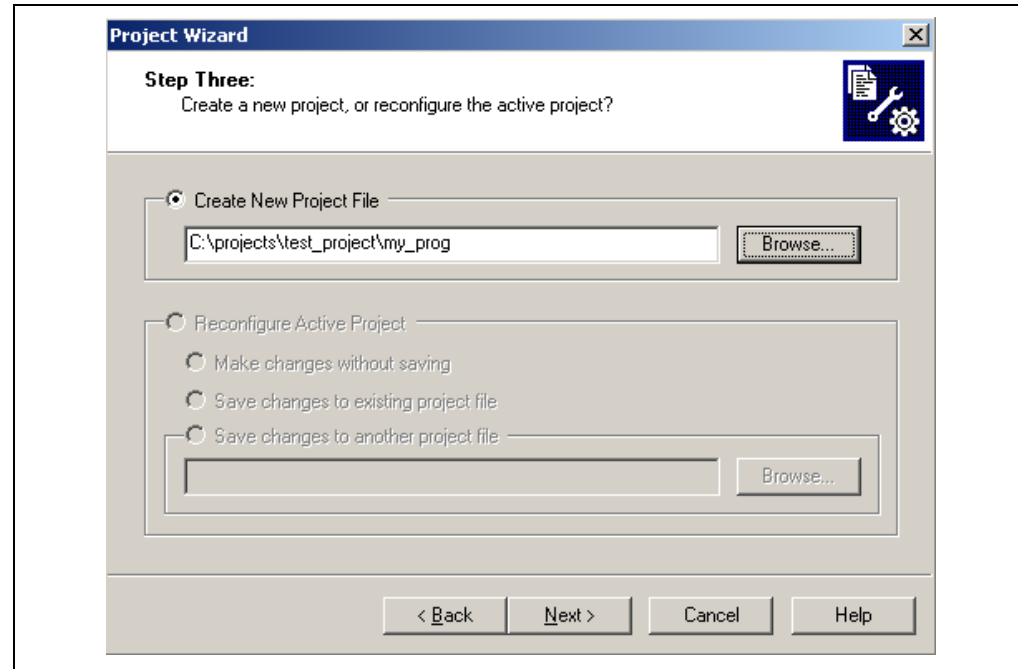


A.3 STEP THREE: CREATE NEW PROJECT FILE

This step creates the actual project file including the path to that project. Ensure that the radio box “Create New Project File” is selected and then click on the “Browse” button. Browse to the desired location and enter the project name in the “File name” text box. Click on “Save”. This will fill in the full path name in the project name text box. An example is shown in [Figure A-3](#).

Consult your lab instructor on the actual path to the local project folders.

FIGURE A-3: PROJECT WIZARD STEP THREE



A.4 STEP FOUR: ADD EXISTING PROJECT FILES

In this step we'll add any files that already exist for our project. For Lab 0, this includes the following files:

- config_bits_pic24fj256gb110.inc
- p24fj256gb110.gld
- lab0-intro.s

Browse to the default lab folder (see lab instructor for specific details) and highlight these three files and click on the “Add>>” button to add them to the project. The files can be selected individually with a discrete pressing of the “Add>>” button for each file. The files can also be selected at once by holding down the “Ctrl” key on the keyboard and then pressing “Add>>” button once to add all the files at once.

Click on the “Next” button to continue and then click on “Finish” to complete. The project files are created. This includes a file with the extension .mcs that contains the project settings, and a file with the extension .mcw that contains the workspace settings.

For best results, load the workspace and save its state to keep the window locations and other configuration settings consistent between editing sessions.



MICROCHIP

Worldwide Sales and Service

AMERICAS

Corporate Office
2355 West Chandler Blvd.
Chandler, AZ 85224-6199
Tel: 480-792-7200
Fax: 480-792-7277
Technical Support:
<http://www.microchip.com/support>
Web Address:
www.microchip.com

Atlanta
Duluth, GA
Tel: 678-957-9614
Fax: 678-957-1455

Boston
Westborough, MA
Tel: 774-760-0087
Fax: 774-760-0088

Chicago
Itasca, IL
Tel: 630-285-0071
Fax: 630-285-0075

Cleveland
Independence, OH
Tel: 216-447-0464
Fax: 216-447-0643

Dallas
Addison, TX
Tel: 972-818-7423
Fax: 972-818-2924

Detroit
Farmington Hills, MI
Tel: 248-538-2250
Fax: 248-538-2260

Indianapolis
Noblesville, IN
Tel: 317-773-8323
Fax: 317-773-5453

Los Angeles
Mission Viejo, CA
Tel: 949-462-9523
Fax: 949-462-9608

Santa Clara
Santa Clara, CA
Tel: 408-961-6444
Fax: 408-961-6445

Toronto
Mississauga, Ontario,
Canada
Tel: 905-673-0699
Fax: 905-673-6509

ASIA/PACIFIC

Asia Pacific Office
Suites 3707-14, 37th Floor
Tower 6, The Gateway
Harbour City, Kowloon
Hong Kong
Tel: 852-2401-1200
Fax: 852-2401-3431

Australia - Sydney
Tel: 61-2-9868-6733
Fax: 61-2-9868-6755

China - Beijing
Tel: 86-10-8569-7000
Fax: 86-10-8528-2104

China - Chengdu
Tel: 86-28-8665-5511
Fax: 86-28-8665-7889

China - Chongqing
Tel: 86-23-8980-9588
Fax: 86-23-8980-9500

China - Hangzhou
Tel: 86-571-2819-3180
Fax: 86-571-2819-3189

China - Hong Kong SAR
Tel: 852-2401-1200
Fax: 852-2401-3431

China - Nanjing
Tel: 86-25-8473-2460
Fax: 86-25-8473-2470

China - Qingdao
Tel: 86-532-8502-7355
Fax: 86-532-8502-7205

China - Shanghai
Tel: 86-21-5407-5533
Fax: 86-21-5407-5066

China - Shenyang
Tel: 86-24-2334-2829
Fax: 86-24-2334-2393

China - Shenzhen
Tel: 86-755-8203-2660
Fax: 86-755-8203-1760

China - Wuhan
Tel: 86-27-5980-5300
Fax: 86-27-5980-5118

China - Xian
Tel: 86-29-8833-7252
Fax: 86-29-8833-7256

China - Xiamen
Tel: 86-592-2388138
Fax: 86-592-2388130

China - Zhuhai
Tel: 86-756-3210040
Fax: 86-756-3210049

ASIA/PACIFIC

India - Bangalore
Tel: 91-80-3090-4444
Fax: 91-80-3090-4123

India - New Delhi
Tel: 91-11-4160-8631
Fax: 91-11-4160-8632

India - Pune
Tel: 91-20-2566-1512
Fax: 91-20-2566-1513

Japan - Yokohama
Tel: 81-45-471- 6166
Fax: 81-45-471-6122

Korea - Daegu
Tel: 82-53-744-4301
Fax: 82-53-744-4302

Korea - Seoul
Tel: 82-2-554-7200
Fax: 82-2-558-5932 or
82-2-558-5934

Malaysia - Kuala Lumpur
Tel: 60-3-6201-9857
Fax: 60-3-6201-9859

Malaysia - Penang
Tel: 60-4-227-8870
Fax: 60-4-227-4068

Philippines - Manila
Tel: 63-2-634-9065
Fax: 63-2-634-9069

Singapore
Tel: 65-6334-8870
Fax: 65-6334-8850

Taiwan - Hsin Chu
Tel: 886-3-6578-300
Fax: 886-3-6578-370

Taiwan - Kaohsiung
Tel: 886-7-213-7830
Fax: 886-7-330-9305

Taiwan - Taipei
Tel: 886-2-2500-6610
Fax: 886-2-2508-0102

Thailand - Bangkok
Tel: 66-2-694-1351
Fax: 66-2-694-1350

EUROPE

Austria - Wels
Tel: 43-7242-2244-39
Fax: 43-7242-2244-393

Denmark - Copenhagen
Tel: 45-4450-2828
Fax: 45-4485-2829

France - Paris
Tel: 33-1-69-53-63-20
Fax: 33-1-69-30-90-79

Germany - Munich
Tel: 49-89-627-144-0
Fax: 49-89-627-144-44

Italy - Milan
Tel: 39-0331-742611
Fax: 39-0331-466781

Netherlands - Drunen
Tel: 31-416-690399
Fax: 31-416-690340

Spain - Madrid
Tel: 34-91-708-08-90
Fax: 34-91-708-08-91

UK - Wokingham
Tel: 44-118-921-5869
Fax: 44-118-921-5820

05/02/11